Chapter 5

Signal Descriptions

CHAPTER 5. SIGNAL DESCRIPTIONS

5. SIGNAL DESCRIPTIONS

This section contains a brief description of the signals on the DSP3210. Figure 5-1 illustrates the functional organization of these pins which are divided into six categories: memory interface, serial I/O, bit I/O, utility, interrupt, and power supply. Table 5-1 is a summary of the signal pins provided in the 132 PQFP package.

Table 5-1 Pin Summary

Bus Interface	D0—D31, A2—A31, MS0—MS3	66
DRAM Signals	CSN, RW, PBD/BLMN*	3
SRAM Signals	ASN, MRN, MWN	1†
Control	SRDYN, BERRN, AEN, DEN, DLE	5
Arbitration Signals	BRN, BGN, BGACKN, LOCKN	4
Serial I/O	DI, ICK, ILD, DO, OCK, OLD, SY	7
Bit I/O	BIO0—BIO7	8
Utility	CKI, RESTN, ZN	3
Interrupt	IR0N, IR1N, IACK0, IACK1	4
Vss		20
VDD		11
	Total	132

* PBD(BLMN) - Page Break Detect and Block Move share the same pin, selection is made via pcw[14—15]
† AEN is multiplexed with MRN and DEN is multiplexed with MWN, selection is made via pcw[12]

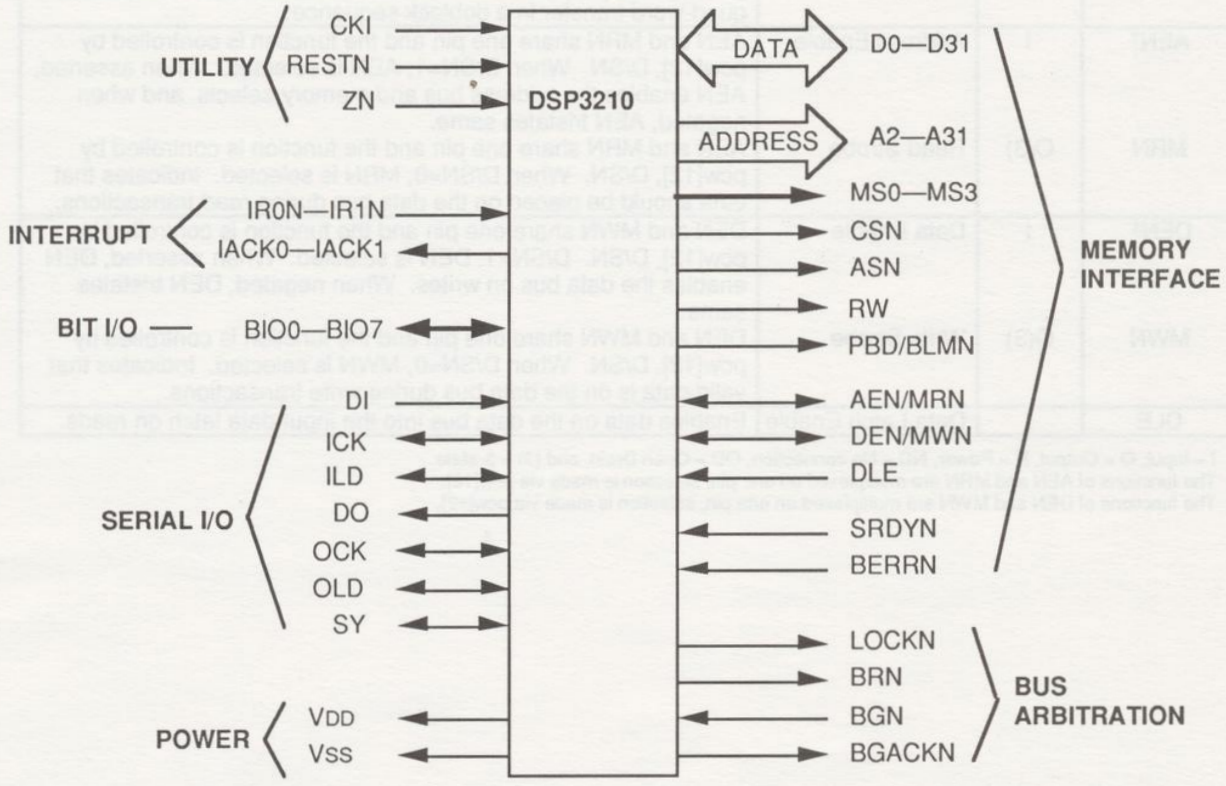


Figure 5-1 DSP3210 Signal Groups

A description of the function of each pin is presented in Table 5-2.

Table 5-2 Pin Descriptions

Pin Name	Type*	Name	Brief Description
D0—D31	I/O(3)	External Memory Data Bus	Bidirectional signals provide the general purpose data path between the DSP3210 and system resources. The data bus transmits and receives 8-, 16-, and 32-bit data.
A2—A31	O(3)	External Memory Address Bus	Provides the word address for DSP bus transfers.
MS0-MS3	O(3)	Memory Selects	Provides memory select information. Signaling based on pcw[9], M/IN: M/IN=1: Memory selects convey data type and word offset. M/IN=0: Memory selects convey byte select information.
CSN	O(3)	Cycle Start	Indicates the beginning of a valid external memory transaction.
ASN	O(3)	Address Strobe	Indicates that address and R/W information are valid.
RW	O(3)	Read/Write	Defines the direction of a data transfer. Signaling based on pcw[11], R/WN: R/WN=1: A high level indicates a read from an external device, a low level indicates a write to an external device. R/WN=0: A low level indicates a read from an external device, a high level indicates a write to an external device.
PBD/BLMN	O(3)	Page Break Detect/Block Move	Provides Page Break Detect or Block Move indication based on pcw[15—14], P/BN. P/BN=01, 1X: Indicates that the current memory access is not on the same page/not sequential as the previous memory access. P/BN=0: Indicates the first word of quad-word transfer in a doblock sequence.
AEN†	I	Address Enable	AEN and MRN share one pin and the function is controlled by pcw[12], D/SN. When D/SN=1, AEN is selected. When asserted, AEN enables the address bus and memory selects, and when negated, AEN tristates same.
MRN	O(3)	Read Strobe	AEN and MRN share one pin and the function is controlled by pcw[12], D/SN. When D/SN=0, MRN is selected. Indicates that data should be placed on the data bus during read transactions.
DEN‡	I	Data Enable	DEN and MWN share one pin and the function is controlled by pcw[12], D/SN. D/SN=1: DEN is selected. When asserted, DEN enables the data bus on writes. When negated, DEN tristates same.
MWN	O(3)	Write Strobe	DEN and MWN share one pin and the function is controlled by pcw[12], D/SN. When D/SN=0, MWN is selected. Indicates that valid data is on the data bus during write transactions.
DLE	I	Data Latch Enable	Enables data on the data bus into the input data latch on reads.

* I = Input, O = Output, P = Power, NC = No connection, OD = Open Drain, and (3) = 3-state.
† The functions of AEN and MRN are multiplexed on one pin, selection is made via pcw[12].
‡ The functions of DEN and MWN are multiplexed on one pin, selection is made via pcw[12].

Table 5-2 Pin Descriptions (continued)

Pin Name	Type*	Name	Brief Description
SRDYN	I	Synchronous Ready	Indicates that a data transfer is complete. When asserted, the DSP3210 completes the current transaction. This is a synchronous input and setup and hold times must be met for reliable operation. SRDYN is used in conjunction with BERRN and BGN to control relinquish, retry, and bus error operations.
BERRN	I	Bus Error	Indicates that a data transfer has encountered a bus error. It is used in conjunction with SRDYN and BGN to control relinquish, retry, and bus error operations.
LOCKN	O(3)	Bus Lock	Asserted when the DSP3210 is requesting interlocked memory transactions.
BRN	O(3)	Bus Request	Asserted when the DSP3210 is requesting mastership of the bus. It indicates that access to external memory is needed.
BGN	I	Bus Grant	Grant for ownership of the bus.
BGACKN	O(3)	Bus Grant Acknowledge	Indicates that the DSP3210 has taken ownership of the data and address busses and control signals. When negated, it indicates that the DSP has placed data, address and control signals in the high-impedance state.
BIO0-BIO7	I/O(3)	Bit I/O	Eight bidirectional signals that can be used for status and control information. Configured by **bioc** register, and data is transferred via **bio** register.
DI	I	Serial Data In	Serial data input.
ICK	I/O(3)	Input Clock	Clock for serial data. In internal mode, ICK is generated on-chip (as specified in **ioc**) and is an output; in external mode, ICK is an input.
ILD	I/O(3)	Input Load	Signal for loading input buffer from internal shift register. In internal mode, ILD is generated on-chip (as specified in **ioc**) and is an output; in external mode, ILD is an input.
DO	O(3)	Serial Data Out	Serial data output.
OCK	I/O(3)	Output Clock	Clock for serial data. In internal mode, OCK is generated on-chip (as specified in **ioc**) and is an output; in external mode, OCK is an input.
OLD	I/O(3)	Output Load	Signal for loading input buffer from internal shift register. In internal mode,OLD is generated on-chip (as specified in **ioc**) and is an output; in external mode, OLD is an input.
SY	I/O(3)	Synchronization	Provides frame synchronization to or from the serial port.
CKI	I	Clock In	System clock.
RESTN	OD	Reset	Highest priority exception. When asserted, the processor enters into the reset state. A low-to-high transition initiates the reset sequence, call r22+0 (r20); nop.
ZN	I	3-state	When asserted, all DSP3210 output pins are 3-stated. ZN has an on-chip pull-up, therefore, it is not necessary that it be connected.
IR0N	I	Interrupt Request 0	Higher priority external interrupt. Maskable in the **emr** register.
IACK0	O(3)	Interrupt Acknowledge 0	Indicates the servicing of the external interrupt request 0.
IR1N	I	Interrupt Request 1	Lower priority external interrupt. Maskable in the **emr** register.
IACK1	O(3)	Interrupt Acknowledge 1	Indicates the servicing of the external interrupt request 1.
V_{DD}	P	Power	
V_{SS}	P	Ground	

* I = Input, O = Output, P = Power, NC = No connection, OD = Open Drain, and (3) = 3-state.

Table 5-3 is a summary of the characteristics of the signals.

Table 5-3 Signal Summary

Signal Function	Signal Name	Input/Output	Active State	3-Stated By			
				ZN	AEN	DEN	BGACKN
Data Bus	D0—D31	Input/Output	High/Low	Yes	—	Yes	—
Address Bus	A2—A31	Output	High/Low	Yes	Yes	—	Yes
Memory Selects	MS0—MS3[1]	Output	High/Low	Yes	Yes	—	Yes
	MS0—MS3[2]	Output	Low	Yes	Yes	—	Yes
Cycle Start	CSN	Output	Low	Yes	—	—	Yes
Address Strobe	ASN	Output	Low	Yes	—	—	Yes
Read/Write	R/W	Output	High/Low	Yes	—	—	Yes
Page Break Detect	PBD	Output	High	Yes	—	—	Yes
Address Enable	AEN[3]	Input	Low	—	—	—	—
Read Strobe	MRN[4]	Output	Low	Yes	—	—	Yes
Data Enable	DEN[5]	Input	Low	—	—	—	—
Write Strobe	MWN[6]	Output	Low	Yes	-	-	Yes
Data Latch Enable	DLE	Input	High	—	—	—	—
Synchronous Ready	SRDYN	Input	Low	—	—	—	—
Bus Error	BERRN	Input	Low	—	—	—	—
Bus Lock	LOCKN	Output	Low	Yes	—	—	Yes
Bus Request	BRN	Output	Low	Yes	—	—	—
Bus Grant	BGN	Input	Low	—	—	—	—
Bus Grant Acknowledge	BGACKN	Output	Low	Yes	—	—	—
Bit I/O	BIO0—BIO7	Input/Output	High/Low	Yes	—	—	—
Serial Data In	DI	Input	High/Low	—	—	—	—
Serial Input Clock	ICK	Input/Output	——	Yes	—	—	—
Serial Input Load	ILD	Input/Output	High–to–Low	Yes	—	—	—
Serial Data Out	DO	Output	High/Low	Yes	—	—	—
Serial Output Clock	OCK	Input/Output	-	Yes	—	—	—
Serial Output Load	OLD	Input/Output	High–to–Low	Yes	—	—	—
Serial Synchronization	SY	Input/Output	High–to–Low	Yes	—	—	—
Clock In	CKI	Input	—	—	—	—	—
Reset	RESTN	Input/Output	Low	—	—	—	—
3-state	ZN	Input	Low	—	—	—	—
Interrupt Request 0	IR0N	Input	Low	—	—	—	—
Interrupt Acknowledge 0	IACK0	Output	High	Yes	—	—	—
Interrupt Request 1	IR1N	Input	Low	—	—	—	—
Interrupt Acknowledge 1	IACK1	Output	High	Yes	—	—	—
Power	VDD	Input	—	—	—	—	—
Ground	VSS	Input	—	—	—	—	—

1. The function of these pins in based on pcw[9]. This entry is for pcw[9] = 0.
2. The function of these pins in based on pcw[9]. This entry is for pcw[9] = 1.
3. The function of this pin in based on pcw[12]. This entry is for pcw[12] = 1.
4. The function of this pin in based on pcw[12]. This entry is for pcw[12] = 0.
5. The function of this pin in based on pcw[12]. This entry is for pcw[12] = 1.
6. The function of this pin in based on pcw[12]. This entry is for pcw[12] = 0.

Chapter 6

Bus Interface

CHAPTER 6. BUS INTERFACE

6. BUS INTERFACE

This section provides a functional description of the bus, the signals that control it, and the bus cycles provided for data transfer operations and bus arbitration. For exact timing specifications, refer to Section 12 - Electrical Characteristics.

The DSP3210 allows 8-bit, 16-bit, and 32-bit operands to be located in external memory that is typically 32-bits wide. Section 3.4 Memory Organization describes how data of various sizes may be addressed. The DSP3210 has an address bus that specifies the word address for the transfer and a data bus that transfers the data. Control signals indicate the beginning of the cycle, the data size or bytes of a 32-bit word requested, and the type of transfer, read or write. Bus cycles are synchronous. The ready line, SRDYN, the bus error signal, BERRN, and the bus grant signal, BGN, must meet setup and hold times for reliable operation.

6.1 Bus Interface Signal Descriptions

Name	Description
A2—A31	**Address Bus** (Output; 3-state). The address bus conveys word addresses to the external system. The address space of the DSP3210 is partitioned between on-chip memory and external memory. External addresses initiate bus cycles and are accompanied by control strobes. Internal memory addresses are issued externally, but do not initiate bus cycles, unless 1) an external bys cycle is in progress or 2) the DSP3210 is not granted its bus. Only addresses for external memory are accompanied by control strobes. 3-stated by ZN, AEN, and 3-stated when BGACKN is negated.
D0—D31	**Data Bus** (Bidirectional; 3-state). The external data bus transmits data to and from the DSP3210. The data bus can transmit bytes (8-bit data), halfwords (16-bit data), and words (32-bit data). On write transactions, byte data is replicated on the three remaining bytes of the data bus while halfword data is replicated on the remaining halfword of the data bus. 3-stated by ZN and DEN.
MS0—MS3	**Memory Selects** (Output; 3-state). The information transmitted on MS0—MS3 is controlled by two bits in the PCW register, PCW[9—8]. PCW[9], M/IN, controls whether Motorola or Intel style signaling is used to convey the data type and word offset information. PCW[8], B/LN, controls the byte ordering used when addressing internal or external memory. If M/IN = 1, Motorola semantics are selected and MS0—MS3 have the same meaning as A0-A1 and SIZ0-SIZ1, where MS0 is A0, MS1 is A1, MS2 is SIZ0, and MS3 is SIZ1. If M/IN = 0 and B/LN = 0, Intel semantics are selected and MS0—MS3 have the same meaning as BE0#—BE3#, where MS0 is BE0#, MS1 is BE1#, MS2 is BE2#, and MS3 is BE3#. Figure 6-1 shows the signal information for all data accesses supported by the DSP3210. Also shown is the signaling supplied if M/IN = 0 and B/LN=1. 3-stated by ZN, AEN and 3-stated when BGACKN is negated.
CSN	**Cycle Start** (Output; Active low; 3-state). When asserted (low), this signal indicates the beginning of an external memory transaction. CSN may be used in DRAM memory designs to generate an early RAS signal. 3-stated by ZN and 3-stated when BGACKN is negated.
ASN	**Address Strobe** (Output; Active low; 3-state). When asserted (low), this signal indicates that the address (A2—31) and memory selects (MS0—MS3) are valid. When negated, this signal indicates that the memory transaction is complete. 3-stated by ZN and 3-stated when BGACKN is negated.
RW	**Read/Write** (Output; 3-state). The sense of the read/write signal is controlled by the R/WN bit in the PCW register, PCW[11]. If R/WN = 1, RW has the same semantics as R/W- in a Motorola 680x0. When high, a read transaction is indicated, and when low, a write transaction is indicated. If R/WN = 0, RW has the same semantics as W/R# in an Intel i386. When high, a write transaction is indicated, and when low, a read transaction is indicated. 3-stated by ZN and 3-stated when BGACKN is negated.

Bus Interface Signal Descriptions (continued)

Name	Description
LOCKN	**Bus Lock** (Output; Active Low; 3-state). When asserted (low), this signal indicates that the DSP3210 is requesting interlocked bus transactions. The LOCKN is asserted during the execution of a dolock instruction sequence. 3-stated by ZN and 3-stated when BGACKN is negated.
BLMN	**Block Move** (Output; Active Low; 3-state) BLMN and PBD share one pin. When pcw[15—14]=00, BLMN is selected. When asserted [low], BLMN indicates that the DSP3210 is starting to perform a quad-word block move operation. The BLMN is asserted each time during the **doblock** instruction sequence in which the two least significant word address lines, A2—A3, are 0 and at least four words remain to be transferred. 3-stated by ZN and 3-stated when BGACKN is negated.
PBD	**Page Break Detect** (Output; Active high; 3-state). BLMN and PBD share one pin. When pcw[15—14] = 01, 10, or 11, PBD is selected. When asserted [high], this signal indicates that the current external memory access is not on the same page as the previous external memory access. The page size is specified in the pcw register, pcw[4—6], and the default setting is 256 words. When a page size of 256 words is selected, sequential addressing break detection can also be enabled by pcw[7]. Used to achieve higher speed accesses to static column and page mode DRAMs or VRAMs. PBD is always asserted for the first transaction after the DSP3210 takes ownership of the bus. 3-stated by ZN and 3-stated when BGACKN is negated.

Data Type	Word Offset	Data Type Memory Selects												DSP3210 Write Data			
		M/IN=1 & B/LN=X				M/IN=0 & B/LN=0				M/IN=0 & B/LN=1							
		MS3	MS2	MS1	MS0	MS3	MS2	MS1	MS0	MS3	MS2	MS1	MS0	24—31	16—23	8—15	0—7
8-bit	0	0	1	0	0	1	1	1	0	0	1	1	1	**A**	A	A	A
8-bit	1	0	1	0	1	1	1	0	1	1	0	1	1	B	**B**	B	B
8-bit	2	0	1	1	0	1	0	1	1	1	1	0	1	C	C	**C**	C
8-bit	3	0	1	1	1	0	1	1	1	1	1	1	0	D	D	D	**D**
16-bit	0	1	0	0	0	1	1	0	0	0	0	1	1	**B**	**A**	B	A
16-bit	2	1	0	1	0	0	0	1	1	1	1	0	0	D	C	**D**	**C**
32-bit	0	0	0	0	0	0	0	0	0	0	0	0	0	**D**	**C**	**B**	**A**
4-word	0	1	1	0	0	-†	-	-	-	-	-	-	-	**D**	**C**	**B**	**A**

† 4-word transfers are also signaled with the BLMN pin.

Write Data Key:

A = logical write data D0—7 with little endian byte ordering.
B = logical write data D8—15 with little endian byte ordering.
C = logical write data D16—23 with little endian byte ordering.
D = logical write data D24—31 with little endian byte ordering.

Underline indicates logical position for write data for little endian byte ordering.
Bold indicates logical position for write data for big endian byte ordering.

Figure 6-1 External Memory Select Signaling and Write Data Duplication

Bus Interface Signal Descriptions (continued)

Name	Description
AEN	**Address Enable** (Input; Active Low). AEN and MRN share one pin. When pcw[12] = 1, AEN is selected. When asserted (low), AEN enables the address bus, A2—31, and memory selects, MS0—MS3, and when negated, AEN tristates same (Note that ZN and BGACKN take precedence over AEN). AEN, along with DEN, can be used to multiplex the address bus and data bus.
MRN	**Read Strobe** (Output; Active low; 3-state). AEN and MRN share one pin. When pcw[12] = 0, MRN is selected. MRN is asserted (low) during read transactions to indicate that data may be placed on the data bus (D0—D31). MRN is typically used in SRAM memory designs as an output enable. 3-stated by ZN and 3-stated when BGACKN is negated.
DEN	**Data Enable** (Input; Active Low). DEN and MWN share one pin. When pcw[12] = 1, DEN is selected. When asserted (low), this input enables the data bus (D0—D31) during write transactions (Note that ZN takes precedence over DEN). DEN, along with AEN, can be used to multiplex the address bus and data bus.
MWN	**Write Strobe** (Output; Active low; 3-state). DEN and MWN share one pin. When pcw[12] = 0, MWN is selected. MWN is asserted (low) during write transactions and data setup and hold times are provided around its rising edge. MWN is typically used in SRAM memory designs as a write enable. 3-stated by ZN and 3-stated when BGACKN is negated.
SRDYN	**Synchronous Ready** (Input; Active Low). Indicates that a synchronous data transfer is complete. When asserted, the DSP3210 completes the current transaction. This is a synchronous input and setup and hold times with respect to CKI must be met for reliable operation. SRDYN is used in conjunction with BERRN and BGN to signal bus errors, retry, and relinquish-retry operations.
BERRN	**Bus Error** (Input; Active Low). Indicates that the data transfer is invalid. It is used in conjunction with SRDYN and BGN to indicate that a retry, relinquish/retry, or bus error exception should be taken. A bus error causes the processor to proceed to an exception handler. Refer to Section 6.4 - Bus Error for the encoding and assertion results for SRDYN, BGN and BERRN. This is a synchronous input and setup and hold times with respect to CKI must be met for reliable operation.
DLE	**Data Latch Enable** (Input; Active High). When asserted (high), this input enables data on the data bus (D0—D31) into the input latch. DLE is provided to simplify the design for a DSP3210 communicating on a system bus operating asynchronously from the DSP3210.

6.2 Processor Control Word

The Processor Control Word, pcw, is a 16-bit internal register that can be read or written using selected CA Data Move Instructions, such as pcw = r1. The pcw register is used to configure the external memory wait-states in partitions A and B, the page size for generating the page break detect (PBD), the memory map, the byte ordering, and the signaling supplied for external memory interfacing. Figure 6-2 shows the bit encoding of the pcw register. By writing a one in Bit 13 with a program, the value loaded into the pcw register can be locked so that programs do not inadvertently change its value. The effect of writing this bit is cleared when the processor is reset or when an error exception is taken. pcw[10—13] are loaded with the logic values applied to pins BIO[7:4], respectively. The remaining pcw register bits are set or cleared by reset as shown.

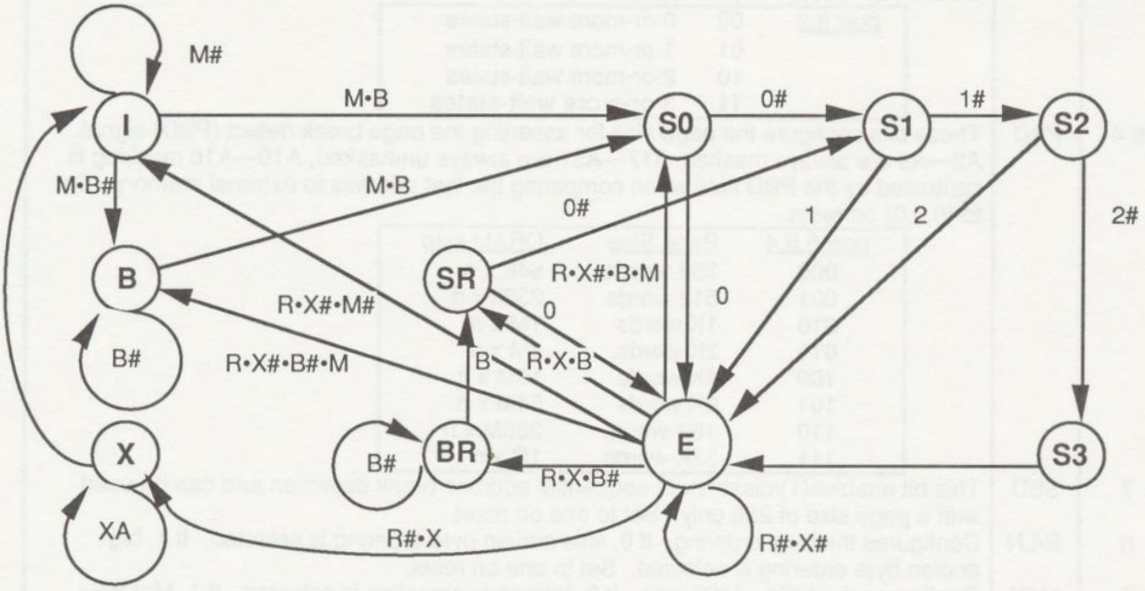
Bit	15 14	13	12	11	10	9	8	7	6 5 4	3 2	1 0
Field	P/BN	BRC	D/SN	R/WN	C/PN	M/IN	B/LN	SBD	PBD	WB	WA

<table>
<tr><th>Bit(s)</th><th>Field</th><th>Function</th></tr>
<tr><td>1,0</td><td>WA</td><td>These bits configure the number of wait-states that are generated for transactions in partition A. Set to {1,1} on reset.
pcw 1,0 — 00 0-or-more wait-states; 01 1-or-more wait-states; 10 2-or-more wait-states; 11 3-or-more wait-states</td></tr>
<tr><td>3,2</td><td>WB</td><td>These bits configure the number of wait-states that are generated for transactions in partition B. Set to {1,1} on reset.
pcw 3,2 — 00 0-or-more wait-states; 01 1-or-more wait-states; 10 2-or-more wait-states; 11 3-or-more wait-states</td></tr>
<tr><td>6-4</td><td>PBD</td><td>These bits configure the page size for asserting the page break detect (PBD) signal. A2—A9 are always masked, A17—A31 are always unmasked, A10—A16 masking is controlled by the PBD field when comparing the last address to external memory. Set to {0,0,0} on reset.
pcw 6,5,4 — Page Size — DRAM size
000 — 256 words — 64k x n
001 — 512 words — 256k x n
010 — 1K words — 1M x n
011 — 2K words — 4M x n
100 — 4K words — 16M x n
101 — 8K words — 64M x n
110 — 16K words — 256M x n
111 — 32K words — 1G x n</td></tr>
<tr><td>7</td><td>SBD</td><td>This bit enables(1)/disables(0) sequential address break detection and can be used with a page size of 256 only. Set to one on reset.</td></tr>
<tr><td>8</td><td>B/LN</td><td>Configures the byte ordering. If 0, little endian byte ordering is selected. If 1, big endian byte ordering is selected. Set to one on reset.</td></tr>
<tr><td>9</td><td>M/IN</td><td>Configures the MS0—MS3 pins. If 0, Intel style signaling is selected. If 1, Motorola style signaling is selected. Set to one on reset.</td></tr>
<tr><td>10</td><td>C/PN</td><td>Selects between one of two memory maps. If 1, µComputer mode is selected and on-chip memory is at location zero. If 0, µProcessor mode is selected and external memory is at location zero. Reset value based on the state of bio[7] during reset.</td></tr>
<tr><td>11</td><td>R/WN</td><td>Configures the RW pin. If 0, RW pin is active high for write transactions. If 1, RW pin is active high for read transactions. Reset value based on the state of bio[6] during reset.</td></tr>
<tr><td>12</td><td>D/SN</td><td>Configures the operation of pins AEN/MRN and DEN/MWN. If 0, SRAM interface signaling selected (MRN and MWN). If 1, DRAM system bus interface signaling selected (AEN and DEN). Reset value based on the state of bio[5] during reset.</td></tr>
<tr><td>13</td><td>BRC</td><td>Reset value based on the state of bio[4] during reset. Controls Boot ROM routine. Also, when written with a one, modification of the pcw is disabled.</td></tr>
<tr><td>15—14</td><td>P/BN</td><td>Configures the PBD/BLMN signal. Set to {0,0} on reset.
pcw 15,14 — 00 BLMN is selected
01 PBD is selected; addresses to partition A are monitored.
10 PBD is selected; addresses to partition B are monitored.
11 PBD is selected; addresses to partitions A and B are monitored.</td></tr>
</table>

Figure 6-2 PCW Register Encoding

6.3 Standard Bus Cycles

The DSP3210 inherently supports synchronous bus cycles. Synchronous bus cycles are completed by handshaking using the SRDYN signal. The point in the transaction at which SRDYN is observed can be altered by configuring the pcw register for zero-, one-, two-, or three-or-more wait-states. The number of wait-states inserted is the longer of the number programmed or the number added using handshaking. The operation of the bus is summarized in Figure 6-3. Sections 6.3.1 to 6.3.5 walk through the operation of the standard bus cycles available in the DSP3210. For the discussion, it assumes that bus error (BERRN) is not asserted and that pcw[9] is configured for SRAM signaling (MRN and MWN).



INPUTS		STATES	
0	Zero programmed wait-states	I	Idle State (Initial State)
1	One programmed wait-state	S0	Initial Bus Cycle State
2	Two programmed wait-states	S1	First Programmed Wait-state
3	Three programmed wait-states	S2	Second Programmed Wait-state
M	External Memory request asserted	S3	Third Programmed Wait-state
B	Bus Grant (BGN) asserted	E	End Cycle State
R	Synchronous Ready (SRDYN) asserted	B	Bus Grant State
X	Bus Error (BERRN) asserted	BR	Bus Relinquish State
XA	Error Exception Acknowledge	SR	Retry
•,+,#	Logical AND,OR, and NOT, respectively	X	Exception State

Figure 6-3 Bus Interface State Diagram

6.3.1 Read Transaction (0-or-more waits)

A read transaction configured with 0-or-more wait-states takes a minimum of two clock periods to complete. Each clock period consists of two phases, ø0 and ø1. Wait-states repeat State E (see Figure 6-4).

State S0 (ø0): The address (A2—A31), the memory selects (MS0—MS3), the read/write signal (RW), and the page break detect (PBD/BLMN) are issued. The cycle start strobe (CSN) is asserted.
State S0 (ø1): The address strobe (ASN), and the read strobe (MRN) are asserted.
State E (ø0): CSN is negated.
State E (ø1): During phase 1, the synchronous ready line (SRDYN) is observed to indicate the end of the transfer. If SRDYN is high, a wait-state is inserted, and if it is low, the transfer is completed. At the end of the transfer, the DSP3210 samples the data bus (D0—D31), and ASN and MRN are negated.

6.3.2 Synchronous Write Transaction (0-or-more waits)

A write transaction configured with 0-or-more wait-states takes a minimum of two clock periods to complete. Each clock period consists of two phases, ø0 and ø1. Wait-states repeat State E (see Figure 6-4).

State S0 (ø0): The address (A2—A31), the memory selects (MS0—MS3), the read/write signal (RW), and the page break detect (PBD/BLMN) are issued. The cycle start strobe (CSN) is asserted.
State S0 (ø1): The address strobe (ASN), and the write strobe (MWN) are asserted.
State E (ø0): The CSN is negated, and data is enabled onto the external data bus (D0—D31).
State E (ø1): MWN is negated. During ø1, the synchronous ready line (SRDYN) is observed to indicate the end of the transfer. If SRDYN is high, a wait-state is inserted, and if it is low, the transfer is completed. At the end of the transfer, the DSP3210 3-states the data bus (D0—D31), and ASN is negated. Note that MWN is negated at the beginning of State E (ø1) even if wait-states are inserted.

6.3.3 Read Transaction (1 to 3-or-more waits)

A read transaction configured with 1 to 3-or-more wait-states takes a minimum of three clock periods. States S2 and S3 are conditionally needed depending on the number of waits selected. External wait-states repeat State E. (see Figures 6-5 to 6-7).

State S0 (ø0): The address (A2—31), the memory selects (MS0—MS3), the read/write signal (RW), and the page break detect (PBD/BLMN) are issued. The cycle start strobe (CSN) is asserted.
State S0 (ø1): The address strobe (ASN), and the read strobe (MRN) are asserted.
State S1 (ø0):
State S1 (ø1):
State S2 (ø0): Entered only if two or three waits are selected .
State S2 (ø1):
State S3 (ø0): Entered only if three waits are selected.
State S3 (ø1):
State E (ø0): CSN is negated.
State E (ø1): During ø1, the synchronous ready line (SRDYN) is observed to indicate the end of the transfer. If SRDYN is high, a wait-state is inserted, and if it is low, the transfer is completed. At the end of the transfer, the DSP3210 samples the data bus (D0—D31), and ASN and MRN are negated.

6.3.4 Write Transaction (1 to 3-or-more waits)

A write transaction configured with 1 to 3-or-more wait-states takes a minimum of three clock periods to complete. States S2 and S3 are conditionally needed depending on the number of waits selected. External waits repeat State E (see Figures 6-5 to 6-7).

State S0 (ø0): The address (A2—31), the memory selects (MS0—MS3), the read/write signal (RW), and the page break detect (PBD) are issued. The cycle start strobe (CSN) is asserted.

State S0 (ø1): The address strobe (ASN), and the write strobe (MWN) are asserted.

State S1 (ø0):

State S1 (ø1):

State S2 (ø0): Entered only if two or three waits are selected. If two or three waits are configured, data is enabled onto the data bus (D0—D31).

State S2 (ø1):

State S3 (ø0): Entered only if three waits are selected.

State S3 (ø1):

State E (ø0): CSN is negated. if one wait is configured, data is enabled onto the external data bus (D0—D31).

State E (ø1): MWN is negated. During phase 1, the synchronous ready line (SRDYN) is observed to indicate the end of the transfer. If SRDYN is high, a wait-state is inserted, and if it is low, the transfer is completed. At the end of the transfer, the DSP3210 3-states the data bus (D0—D31), and ASN is negated. Note that MWN is negated at the beginning of State E (ø1) even if additional wait-states are inserted.

6.3.5 Idle State

An idle bus state is one clock period long. All control strobes, CSN, ASN, MRN, and MWN, are in their negated state. The RW signal signifies a read. The PBD retains its state from the previous bus cycle. The address bus A2—31 will reflect the address of internal memory accesses if they were initiated in the previous processor state. The MS0—MS3 signals will be high.

6.4 Bus Error

The DSP3210 bus protocol expects the assertion of SRDYN to terminate a bus cycle. If the bus cycle encounters an error condition, external circuitry can:

—request the DSP to retry the transaction

—defer the retry by causing the processor to relinquish the bus

—terminate the transaction and cause the processor to enter an exception handler

These actions are signaled via the BGN, BERRN, and SRDYN pins. Table 6-1 shows the action taken by the DSP for all signal conditions.

Table 6-1 BGN-BERRN-SRDYN Codes and Results

BGN	BERRN	SRDYN	Result
0	0	0	Retry bus transaction immediately
1	0	0	Defer retry, relinquish bus immediately
x	0	1	Terminate transaction and take bus error exception.
x	1	0	Normal termination.
x	1	1	Insert wait-state.

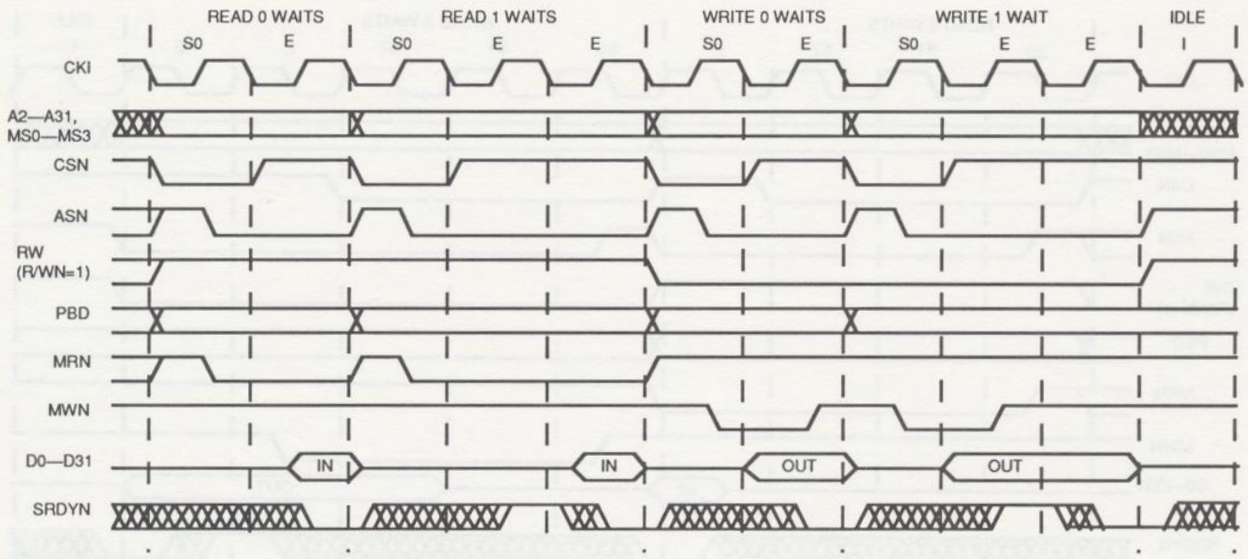


Figure 6-4 Bus Protocol (0-or-more wait-states)

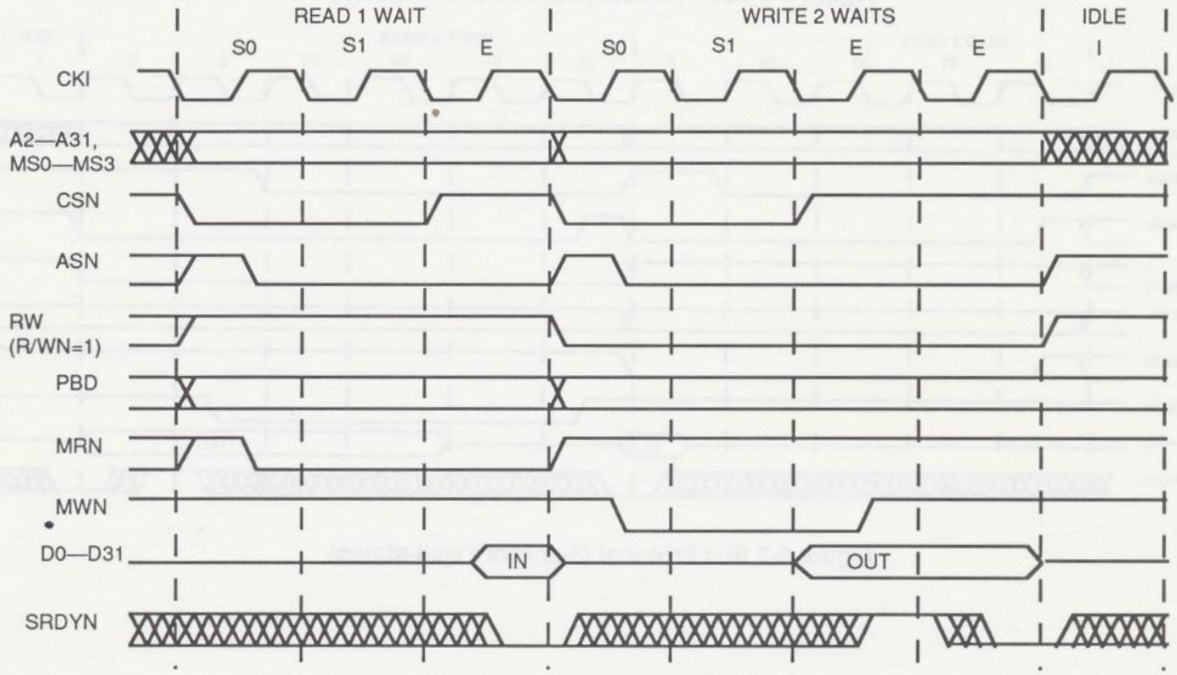


Figure 6-5 Bus Protocol (1-or-more wait-states)

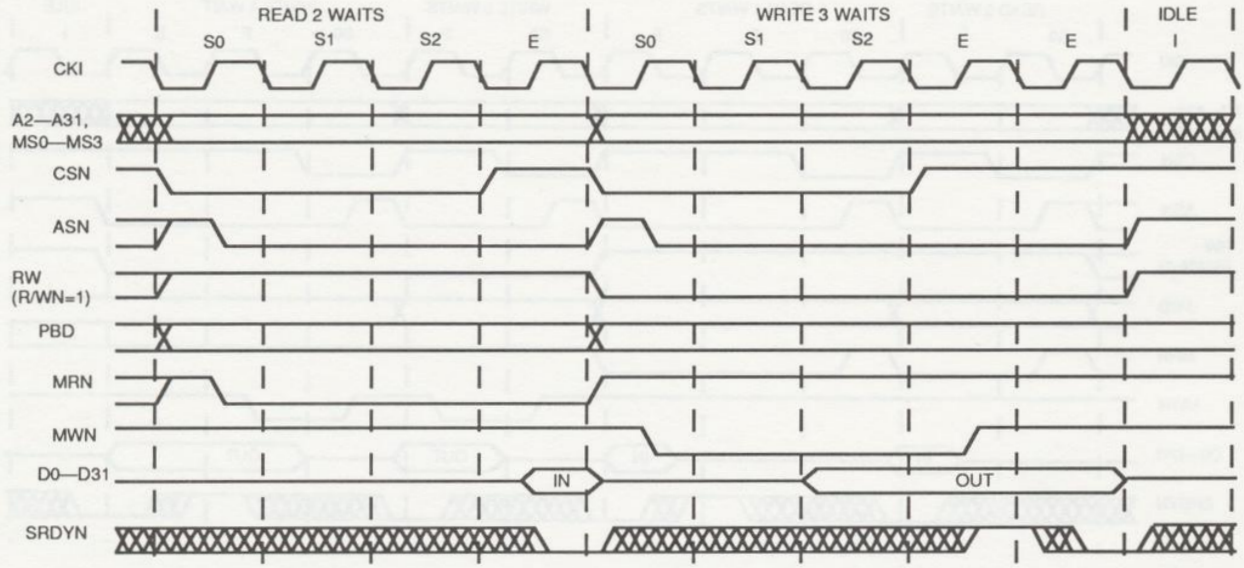


Figure 6-6 Bus Protocol (2-or-more wait-states)

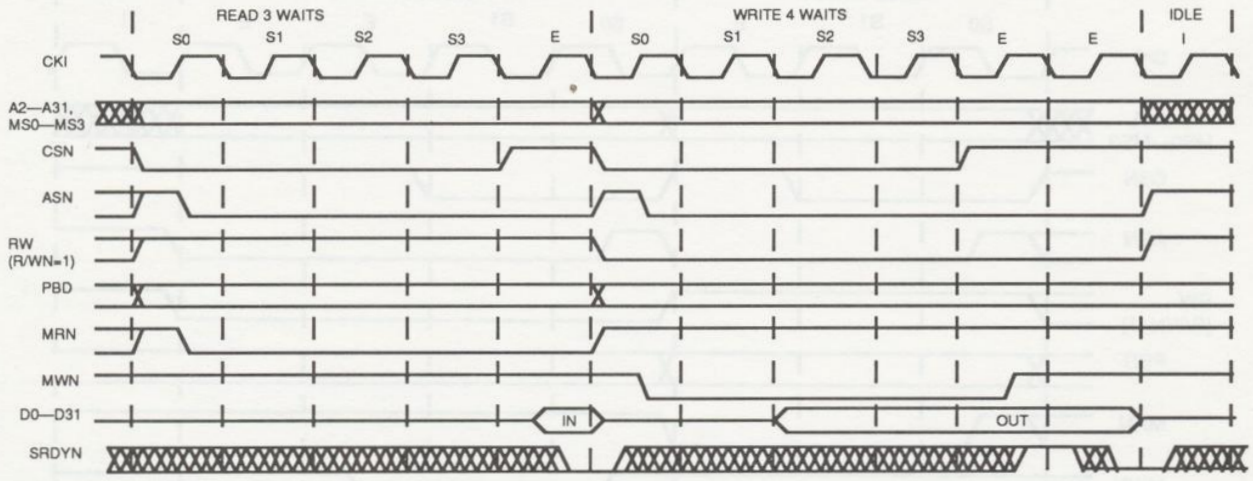


Figure 6-7 Bus Protocol (3-or-more wait-states)

6.4.1 Bus Error Operation

A bus error protocol is signaled to the DSP3210 by the bus error pin, BERRN, in conjunction with SRDYN. The two actions that the DSP can be instructed to take are retry or error exception (stop normal processing and vector to the bus error exception handler, see 7.3 Exception Processing). A retry bus cycle is initiated in the following clock cycle using the same transaction information. Multiple retries can occur if transactions that are retried encounter the retry code when completing. Negating BGN while asserting BERRN and SRDYN provides a relinquish and retry operation where the retry cycle will commence when the DSP regains the bus. When the bus error exception code is detected (BERRN asserted and SRDYN negated), the current bus cycle is terminated, outstanding fetches are invalidated, and the processor proceeds to the exception handler.

Figure 6-8 is a timing diagram of a retry operation.

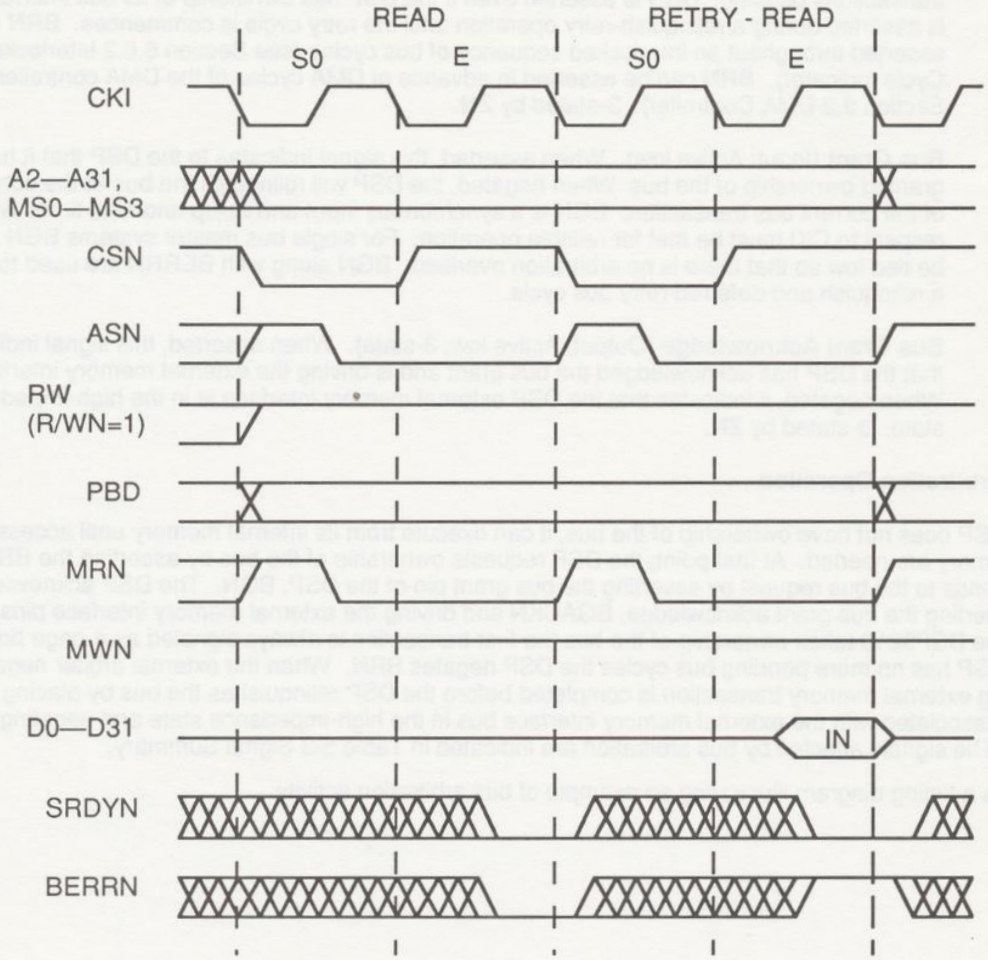


Figure 6-8 Read Cycle Retry

6.5 Bus Arbitration

The DSP3210 shares the external memory interface bus through a request/acknowledge protocol. Bus arbitration is the protocol used by the DSP3210 to become the bus master. An external arbiter controls the arbitration process, and the DSP is a slave device and gains access to the bus by requesting ownership of the bus. The bus arbiter is responsible for assigning priorities to the devices sharing the bus and granting devices the bus.

The three signals associated with the bus arbitration are bus request (BRN), bus grant (BGN), and bus grant acknowledge (BGACKN).

BRN — **Bus Request** (Output; Active low; 3-state). When asserted, this signal indicates that the DSP needs to perform an external memory transaction. BRN is negated when there are no further transactions pending. BRN is asserted even if the DSP has ownership of its bus interface. BRN is asserted during a relinquish-retry operation until the retry cycle is commences. BRN is asserted throughout an interlocked sequence of bus cycles (see Section 6.6.2 Interlocked Bus Cycle Indicator). BRN can be asserted in advance of DMA cycles of the DMA controller (see Section 9.2 DMA Controller). 3-stated by ZN.

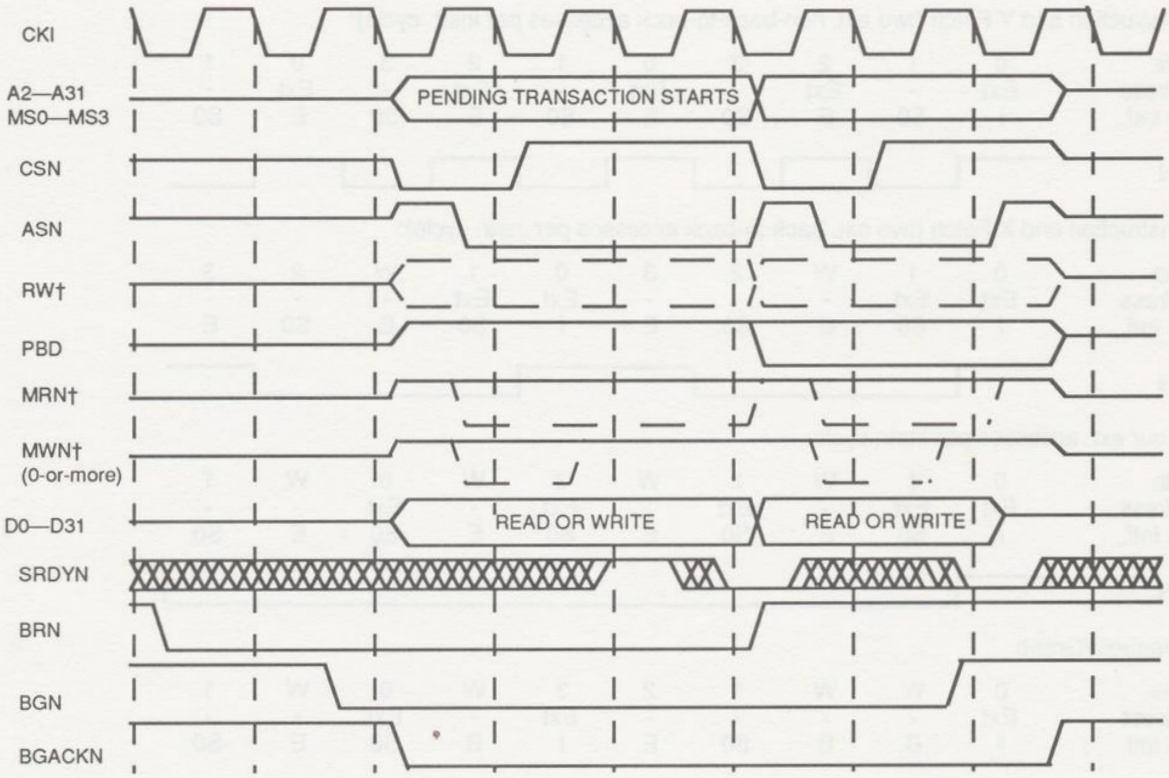
BGN — **Bus Grant** (Input; Active low). When asserted, this signal indicates to the DSP that it has been granted ownership of the bus When negated, the DSP will relinquish the bus at the completion of the current bus transaction. BGN is a synchronous input and setup and hold times with respect to CKI must be met for reliable operation. For single bus master systems BGN should be tied low so that there is no arbitration overlead. BGN along with BERRN are used to support a relinquish and deferred retry bus cycle.

BGACKN — **Bus Grant Acknowledge** (Output; Active low; 3-state). When asserted, this signal indicates that the DSP has acknowledged the bus grant and is driving the external memory interface. When negated, it indicates that the DSP external memory interface is in the high-impedance state. 3-stated by ZN.

6.5.1 Bus Arbitration Operation

When the DSP does not have ownership of the bus, it can execute from its internal memory until accesses to the external memory are needed. At that point, the DSP requests ownership of the bus by asserting the BRN. The arbiter responds to the bus request by asserting the bus grant pin of the DSP, BGN. The DSP acknowledges the grant by asserting the bus grant acknowledge, BGACKN and driving the external memory interface pins. Note that when the DSP3210 takes ownership of the bus the first transaction is always signaled as a page break. When the DSP has no more pending bus cycles the DSP negates BRN. When the external arbiter negates BGN, any on-going external memory transaction is completed before the DSP relinquishes the bus by placing all output and biputs associated with the external memory interface bus in the high-impedance state and negating BGACKN. The signals affected by bus arbitration are indicated in Table 5-3 Signal Summary.

Figure 6-9 is a timing diagram illustrating an example of bus arbitration activity.



† The signal is dependent on whether it is a read or a write cycle.

Figure 6-9 Bus Arbitration Example

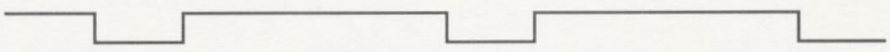
6.5.2 DSP3210 BRN Functional Timing

This section contains examples of BRN functional timing, that is the state of BRN for various sequences of external memory access sequences.

1) Instruction Fetch or doblock (one ext. access per instr. cycle):

State	0	1	2	3	0	1	2	3	0	1
Address	Ext	-	-	-	Ext	-	-	-	Ext	-
Bus Intf.	I	S0	E	I	I	S0	E	I	I	S0

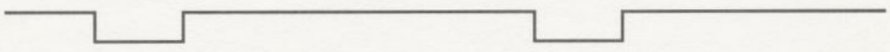
BRN



2) Instruction Fetch or doblock (one ext. access per instr. cycle; one prog. wait):

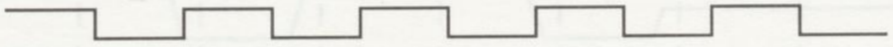
State	0	1	2	W	3	0	1	2	W	3
Address	Ext	-	-	-	-	Ext	-	-	-	-
Bus Intf.	I	S0	S1	E	I	I	S0	S1	E	I

BRN



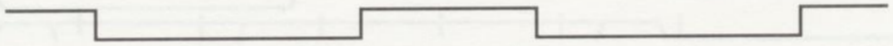
3) Instruction and Y Fetch (two ext. non-back-to-back accesses per instr. cycle):

State	0	1	2	3	0	1	2	3	0	1
Address	Ext	-	Ext	-	Ext	-	Ext	-	Ext	-
Bus Intf.	I	S0	E	S0	E	S0	E	S0	E	S0



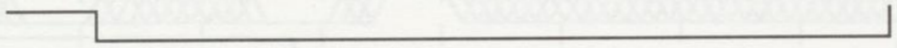
4) Instruction and X Fetch (two ext. back-to-back accesses per instr. cycle):

State	0	1	W	2	3	0	1	W	2	3
Address	Ext	Ext	-	-	-	Ext	Ext	-	-	-
Bus Intf.	I	S0	E	S0	E	I	S0	E	S0	E



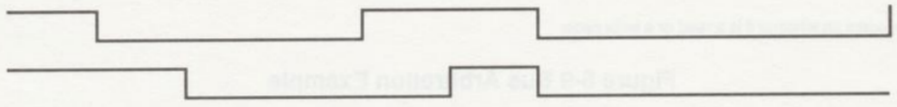
5) Four ext. accesses per instr. cycle:

State	0	1	W	2	W	3	W	0	W	1
Address	Ext	Ext	-	Ext	-	Ext	-	Ext	-	-
Bus Intf.	I	S0	E	S0	E	S0	E	S0	E	S0



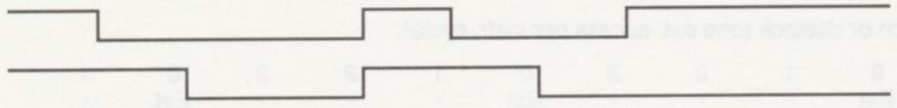
6) Request/Grant:

State	0	W	W	1	2	3	W	0	W	1
Address	Ext	-	-	-	-	Ext	-	Ext	-	-
Bus Intf.	I	B	B	S0	E	I	B	S0	E	S0



7) Relinquish/Retry:

State	0	W	W	1	2	W	W	W	W	3
Address	Ext	-	-	-	-	-	-	-	-	-
Bus Intf.	I	B	B	S0	E	BR	BR	SR	E	I



6.6 Special Interface Capabilities

The DSP3210 has special features that increase system design flexibility. These features improve memory access performance and minimize the number of components necessary when the DSP3210 is being used on an asynchronous and/or multiplexed system bus. The features are implemented with the pins PBD/BLMN, LOCKN, AEN, DEN, and DLE. AEN, DEN, and DLE operate asynchronously with respect to the rest of the external memory interface. Descriptions of these signals is provided in Section 6-1 Bus Interface Signal Descriptions.

6.6.1 Page Break Detect Indicator

The page break detect signal can be used to increase system performance in systems using DRAMs or VRAMs. Bits 4—7 and 14—15 of the pcw register are used to configure the page break detect signal. Bits 4—6 define the page size of the memory device. See Figure 6-2 PCW Register for page size settings. Page sizes can be selected that correspond to DRAMs from 64K to 1G and VRAMs with a 256 bit shift register.

Bits 14—15 define the address space for which page break detection is desired. Partition A, Partition B, or the entire external memory space can be selected as the region the page break detect is monitoring. Addresses outside the selected region are ignored and have no effect on the page break detect signal. This is useful when program and data are separated in different physical external memory devices and increased performance can be achieved by monitoring accesses to only one of the partitions. The page break pin is asserted when an access to the monitored external memory space is not to the same page as the previous access to the monitored external memory space.

Bit 7 is used when the page size is 256 words to enable detection of sequential addressing breaks. This is useful when using VRAMs where data must be read/written sequentially in order to maximize performance. If the next address is not the next word address, $A_{(N+1)} \neq A_{(N)} + 4$, then a sequential addressing break is detected and the page break pin is asserted.

On reset, the page break detect pin is configured to convey BLMN information. This signal will be negated (logic high) which also can be viewed as signaling a page break until the pcw register is set with the appropriate value. The page break is always asserted when the DSP3210 regains bus mastership even if the access is to the same page as the previous external access. Similarly, the page break detect is asserted on the retry of a relinquish-retry operation.

6.6.2 Interlocked Bus Cycle Indicator

The DSP3210 provides a signal, LOCKN, to indicate that it expects to maintain ownership of the bus for one or more additional bus transactions. The LOCKN signal is asserted during the execution of a **dolock** instruction sequence. The bus request pin, BRN, is also asserted for the duration of the dolock instruction sequence. This can be used to perform secure semaphore updates in shared memory systems by emulating a read-modify-write operation within the **dolock** sequence. These instruction sequences are typically not repeated but the structure does allow for it. Interrupts and DMA are disabled during the dolock sequence.

Up to 128 in-line instructions can be encompassed by the dolock indicators, therefore complex and arbitrary bus interlocked sequences can be performed at an overhead of one instruction cycle to execute the dolock instruction. The LOCKN indicator is asserted during the Operand Fetch cycle of the instruction pipeline (see Figure 7-1A Pipeline for a Single DA Instruction). It is not intended that DA instructions with Z-field writes be used in interlocked sequences because the LOCKN indicator will be negated before the write is performed, but four nops following the last DAU write will accomodate their use. Note that CA instruction read and write operations are always generated during the Operand Fetch cycle.

In the following example, instructions I1 to I3 are used to perform a test and set operation. Instructions I4 and I5 are subsequent instructions that cause the processor to branch.

```
I1    dolock 1,0          /* Assert LOCKN and disable interrupts and DMAs for the
                             following two instructions Operand Fetch cycles */
I2    r1=*r2              /* Loading r1 will set the n and z flags for the operand pointed
                             to by r2 */
I3    *r2=r3              /* r3 contains the flag value  to be written back */
I4    if(eq) pcgoto czero /* Flags corresponding to the value in r1 can be tested here */
```

I5 nop

If both instruction and operand accesses were to external memory, the activity and the extent of the assertion of the LOCKN pin for the above sequence of instructions would be:

Access	LOCKN	Comment
I1	1	Fetch dolock 1,0
I2	1	Fetch r1=*r2
I3	1	Fetch *r2=r3
OI2	0	Read operand for I2, *r2
I4	0	Fetch I4
OI3	0	Write operand for I3, *r2
I5	1	Fetch I5

6.6.3 Quad-word Block Move Indicators

A block move is a programmed transfer of a block of data from one location in memory to another location in memory. It is used to move blocks of data from external memory to the DSP3210 internal memory or from the DSP3210 internal memory to external memory. The DSP3210 provides two methods of signaling a quad-word block move operation. When Motorola signaling is selected (pcw[9] = 1), a quad-word transfer is signaled the same as a 68040 burst-inhibited line read. The first word of a group of four words to be transferred sets the MS3—MS2 pins to 11. The remaining three words to be transferred set the MS3—MS2 pins to 00. Secondly, the BLMN pin will be asserted for the first word of the group of four words when selected. The PBD/BLMN pin is selected to be BLMN by setting pcw[15—14] = 00. It is the responsibility of the external bus arbiter to grant the bus to the DSP3210 for the duration of the quad-word transfer.

The instruction sequence used to perform block moves is a **doblock** instruction followed by the DA instruction, **a0=(Z=Y)*a0**, in which the Y operand is written back to memory performing one memory-to-memory transfer per instruction cycle. The doblock loop is decomposed into atomic four word transfers that begin with an address with A3—A2 = 00 and sequence 01, 10, 11. When the do-loop count is four or more and the address has A3—A2 = 00, the start of an atomic quad-word transfer is detected. Interrupts and DMAs are disabled for the next three instruction cycles so that the quad-word transfer is not disrupted. Interrupts and DMA cycles are arbitrated between quad-word transfers. The address of the first read or write location does not have to be quadword aligned, nor does the number of iterations need to be a multiple of 4. Read or writes prior to the detection of a quad-word transfer are standard long word transactions. Reads or writes at the end of the loop when fewer than four words remain are also standard long word transactions.

Due to the DAU pipeline, when performing block moves **to** external memory, at least the last three words and up to as many as the last six words will not be able to be made into quad-word transfers. This is because the remaining block count is compared to the address during the Operand Fetch cycle of the instruction pipeline (See Figure 7-1A Pipeline for a Single DA Instruction). Also, while the last three words are being written subsequent instructions can be fetched invalidating the sequential addressing nature of a quad-word transfer. A block move to external memory should always be followed by three instructions that do not have Z-writes before a subsequent block move is initiated.

The following doblock loop performs a block move of r3 + 1 words:

```
doblock r3              /* Repeat the following instruction r3 + 1 times */
a0= (*r2++=*r1++)*a0    /* Note that this instruction is cached on-chip; the r1 pointer is to the
                           source, and the r2 pointer is to the destination */
3*nop                   /* Three instructions must be executed between block moves to
                           external memory in order to clear the pipe */
```

Figure 6-10 shows the timing for the bus interface signals for a quad-word transfer from external memory to internal memory. It consists of four consecutive read cycles. Even though the DSP3210 has a bus bandwidth of two words per instructions cycle, when using this type of instruction, it can accept data at one word per instruction cycle . Therefore, two idle states are observed between each bus transaction.

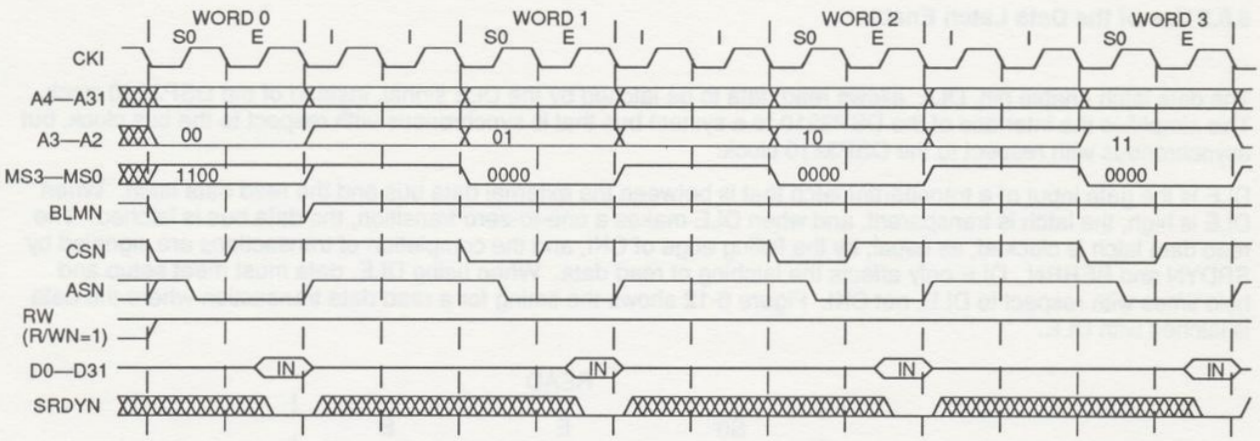


Figure 6-10 Quad-word Block Move Timing

6.6.4 Multiplexed Address and Data Bus

To use the DSP3210 in a multiplexed address and data bus system, AEN and DEN must be used. Basically, an external state machine should control when the address is enabled/disabled for read and write operations, and control when the data is disabled/enabled for write operations. Figure 6-11 shows an example of the timing for a multiplexed address and data bus. AEN and DEN operate asynchronously to the DSP3210 bus protocol. See Section 12 Electrical Characteristics for timing specification for AEN and DEN.

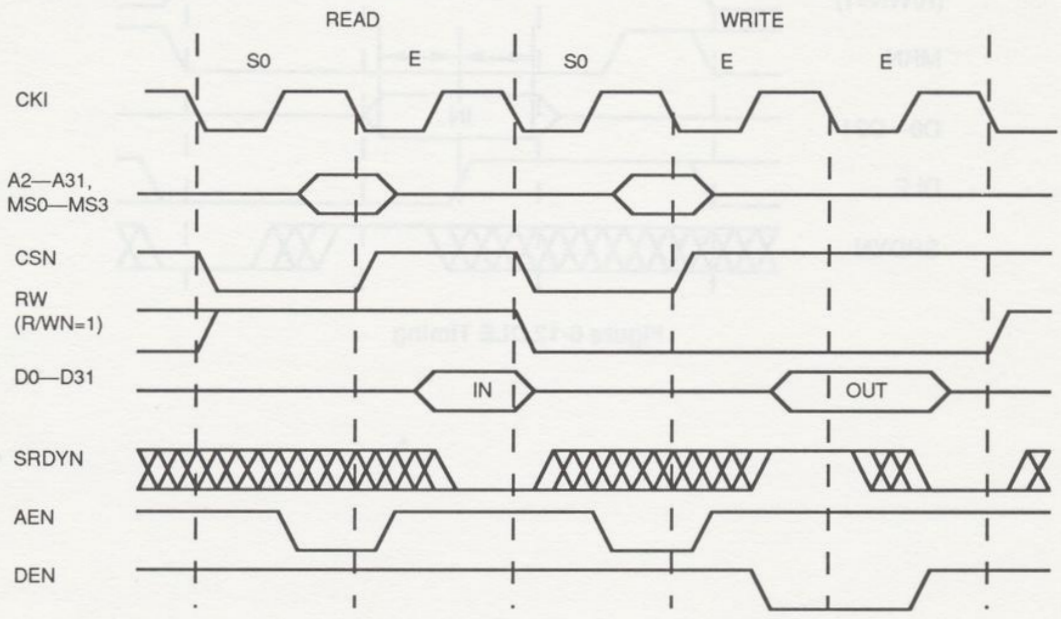


Figure 6-11 Multiplexing the Address Bus and Data Bus

6.6.5 Use of the Data Latch Enable

The data latch enable pin, DLE, allows read data to be latched by the DLE signal, instead of the DSP3210 clock. This simplifies the interface of the DSP3210 to a system bus that is synchronous with respect to the bus clock, but asynchronous with respect to the DSP3210 clock.

DLE is the gate input of a transparent latch that is between the external data bus and the read data latch. When DLE is high, the latch is transparent, and when DLE makes a one-to-zero transition, the data bus is latched. The read data latch is clocked, as usual, by the falling edge of CKI, and the completion of transactions are signaled by SRDYN and BERRN. DLE only affects the latching of read data. When using DLE, data must meet setup and hold times with respect to DLE, not CKI. Figure 6-12 shows the timing for a read data transaction where the data is latched with DLE.

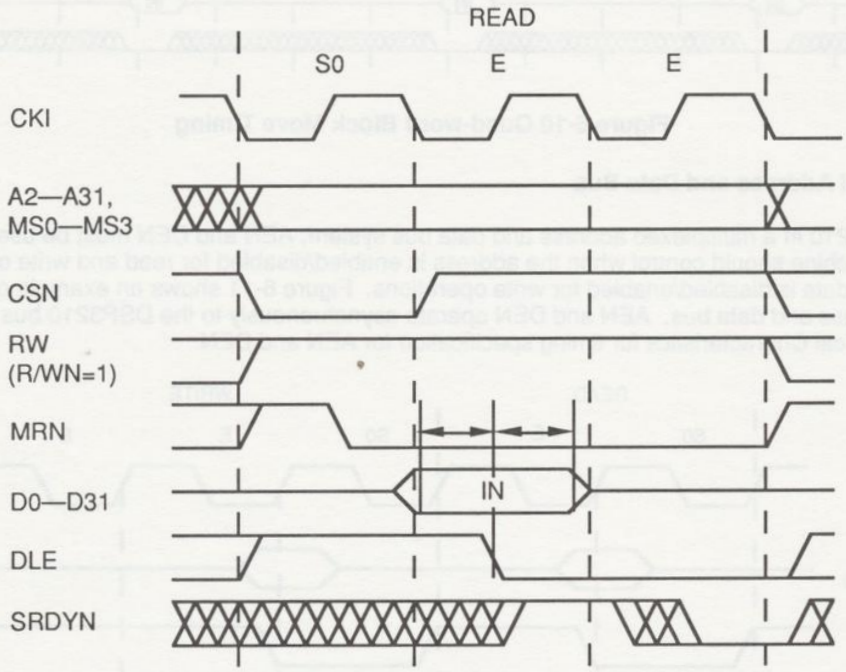


Figure 6-12 DLE Timing

Chapter 7

Processor States

CHAPTER 7. PROCESSOR STATES

7. PROCESSOR STATES

This chapter describes the modes of operation of the DSP3210 . The DSP3210 is equipped with advanced control features that improve performance and simplify system design. This chapter describes the normal instruction cycle, wait-states, direct memory access (DMA), exceptions (including reset, errors, and interrupts), and wait-for-interrupt (powerdown).

7.1 Normal Instruction Cycle

The DSP3210 has an instruction cycle rate of 1/4 the incoming clock frequency (CKI). Each instruction cycle is divided into four machine states, numbered 0-3, where one state equals one period of CKI. In a single instruction cycle, the DSP3210 can perform an instruction fetch, two operand fetches, and a write to memory. The DSP3210 can execute basically two types of instructions; CA instructions and DA instructions (see Section 4 Instruction Set). Each CA instruction can have up to two memory accesses: (1) memory read (**I** instruction), and (2) memory read or write (**X**-read, **Y**-write). Each DA multiply-accumulate instruction can have up to four memory accesses: (1) memory read (**I** instruction), (2) memory read (**X** operand), (3) memory read (**Y** operand), and (4) memory write (**Z** operand). Because the DAU is pipelined, the four accesses do not all occur in the same instruction cycle for a given instruction. Figure 7-1A shows when address and data are on the **internal** address bus and **internal** data bus for one DA instruction. Figure 7-1B illustrates a full pipe of sequential DA multiply-accumulate instructions. The subscripts refer to instruction numbers; instructions are fetched and executed in ascending order. For example, X_0 is the X operand for instruction number 0, and X_{-1} is the X operand for the instruction previous to instruction number 0.

	Instruction Cycle																							
	Instruction Fetch				Instruction Decode				Operand Fetch				Multiply				Accumulate				Store			
State	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
Address Bus	I						X	Y														Z		
Data Bus				I						X	Y										Z			

Figure 7-1A Pipeline for a Single DA Instruction

	Cycle 0				Cycle 1				Cycle 2				Cycle 3				Cycle 4				Cycle 5	
State	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1
Address Bus	I_0		X_{-1}	Y_{-1}	I_1	Z_{-4}	X_0	Y_0	I_2	Z_{-3}	X_1	Y_1	I_3	Z_{-2}	X_2	Y_2	I_4	Z_{-1}	X_3			Z_0
Data Bus				I_0	Z_{-4}	X_{-1}	Y_{-1}	I_1	Z_{-3}	X_0	Y_0	I_2	Z_{-2}	X_1	Y_1	I_3	Z_{-1}	X_2	Y_2	I_4	Z_0	X_3

Figure 7-1B Full Pipeline for Multiple DA Instructions

7.2 Wait-states

The number of states that it takes for an instruction to complete is affected by the introduction of wait-states. Wait-states are introduced when resources required by the instruction are busy and the processor must wait in order to continue processing. There are two types of wait-states: conflict wait-states and external wait-states. Conflict wait-states arise when back-to-back memory requests are made to the same physical memory and the physical memory is either the Boot ROM/MMIO or the external memory. Conflict wait-states **do not** occur when accessing on-chip RAM, RAM0 or RAM1. External wait-states arise when the number of states required to access data in external memory is greater than two or when the DSP3210 is requesting an external memory transaction and it does not have ownership of its bus interface.

The DSP3210 has three physical memories; RAM0/RAM1, External Memory A/B and Boot ROM/MMIO. Some physical memories can introduce conflict wait-states because it takes two states to read or write them and addresses are issued every state. The two physical memories that have this property are the External Memory A/B and the Boot ROM/MMIO. The entire external memory space (Partition A and Partition B) is considered one physical memory, and the combined Boot ROM and Memory-mapped I/O space is considered one physical memory. Wait-states are invisible to the programmer except for their effect of decreasing performance in the DSP3210. However, throughput is degraded only incrementally by the insertion of a conflict wait-state because it is only 1/4 of an instruction cycle.

If external memory is slower than that allowed for zero wait-state operation (2 states), external wait-states are inserted. Also, if the external memory interface is not granted to the processor when an external memory access is required, wait-states are inserted until the bus is granted. Note again that throughput is degraded only incrementally by the insertion of external wait-states because each is only 1/4 of an instruction cycle.

Given the pipelined nature of this device, it is not immediately obvious what successive memory accesses are. Figure 7-1B shows that several DA instructions may be in various stages of execution during a single instruction cycle. To achieve full throughput, no two consecutive accesses can be made to the same physical memory that incurs wait-states. Thus, for example, if the X and Y operands for a given instruction are fetched from the external memory, one conflict wait-state will be inserted between cycle 0-state 3 and cycle 1-state 0. Similarly, if the result for instruction **i** is written to external memory and instruction **i + 4** is fetched from external memory, one conflict wait-state is inserted between cycle 1-state 1, and cycle 1-state 2.

Figure 7-2 shows the effect of conflict and external wait-states on the instruction cycle. In this example, the pipeline is full of DA instructions and all memory accesses are being directed to external memory. Instructions are in logical partition A with zero wait-states, and data is in logical partition B with one wait-state. Conflict wait-states are denoted as Wc, and external wait-states are denoted as WE. Even though there are wait-states, the entire bandwidth of the external memory port is utilized.

	Cycle 0				Cycle 1										
State	0	1	2	3	Wc	WE	0	Wc	WE	1	Wc	2	Wc	WE	3
Address Bus	I_0		X_{-1}	Y_{-1}			I_1			Z_{-4}		X_0		Y_0	
Data Bus				I_0			Z_{-4}			X_{-1}		Y_{-1}		I_1	
External Address Bus		I_0	I_0	X_{-1}	X_{-1}	X_{-1}	Y_{-1}	Y_{-1}	Y_{-1}	I_1	I_1	Z_{-4}	Z_{-4}	Z_{-4}	
External Data Bus			I_0			X_{-1}			Y_{-1}		I_1		Z_{-4}	Z_{-4}	

Figure 7-2 Instruction Cycle with Conflict and External Wait-states

7.3 Direct Memory Access

External devices can access the on-chip memory in the DSP3210, as well as external memory, using direct-memory access (DMA). DMA transfers occur between the memory and the serial I/O port without processor intervention, i.e., using cycle-stealing where the DSP3210 waits for one instruction cycle while the DMA transfer occurs. Two on-chip DMA controllers support memory access via the serial input port and the serial output port.

When DMA is enabled, serial I/O (SIO) transfers occur between the input buffer (ibuf) register and memory and/or between memory and the output buffer (obuf) register without program intervention. The DMA control (dmac) register enables and configures serial I/O DMA activity (see 9.2 DMA Controller). The memory address pointers and the number of transfers to perform for input and output DMA are contained in the input pointer (idp), output pointer (odp), input transfer count (icnt), and output transfer count (ocnt) registers. The serial DMA facility is useful for multichannel applications so that input and output buffers for an entire frame of samples can be established in memory.

Figure 7-3 shows how instruction processing is suspended by the insertion of a DMA cycle. The row labelled IR represents the Instruction Register.

		Cycle 0				DMA Cycle				Cycle 1			
State	3	0	1	2	3	0	1	2	3	0	1	2	3
Address Bus		I_1	Z_{-4}	X_0	Y_0					I_1	Z_{-3}		
Data Bus	I_0	Z_{-4}			I_1		X_0	Y_0		Z_{-3}			I_2
IR		I_0	I_0	I_0	I_0					I_1	I_1	I_1	I_1

Interrupt/DMA Arbitration and Prioritization

DMA Request Synchronization

Figure 7-3 DMA Cycle Timing

7.4 Wait-for-Interrupt Mode

The DSP3210 provides a wait-for-interrupt mode so that the power dissipation of the device is reduced when it is not performing data processing. The wait-for-interrupt mode is enabled by the execution of the instruction **waiti**. This instruction will cause the processor to enter continuous DMA-like cycles. If requested, DMA transfers are performed. When any unmasked interrupt exception is detected, the processor exits the wait-for-interrupt mode and executes the instruction following the waiti instruction before branching to the appropriate interrupt routine (interrupt processing may be initiated only after executing an instruction, not after a DMA cycle). Also, the wait-for-interrupt mode is terminated if an error exception is detected.

7.5 Instruction Sequencing Exceptions

The DSP3210 has four distinct instruction processing levels; Base level, Interrupt level, Error level, and Double Error level. Movement between any level is considered as an exception process. Base level instruction processing can be altered by the introduction of interrupt routines. Base level or Interrupt level instruction processing can be altered by the introduction of error handling routines. Error handling can be altered by the introduction of a double error. Figure 7-4 shows the processing levels in the DSP3210.

The DSP3210 handles interrupts using a continuation exception processing model. Single error conditions are handled using an abort exception processing model. Double error conditions are handled by stopping the DSP3210.

The continuation exception model means that the necessary processor context is saved so that when exception processing is completed, the processor can resume at the point where it left off. The abort exception model means that certain processor context is cleared and the current instruction cycle is aborted. The processor can not resume processing where it left off. Register information to locate the cause of the error is maintained so that debugging can be done (refer to Table 7-3). When a double error condition is recognized, the DSP3210 aborts the current instruction, stops, and signals the external system. At this point, the DSP3210 must be reset to start instruction processing.

Interrupt exception, DMA, and instruction processing decisions are accepted at each instruction cycle boundary. In fact, arbitration is done during state 1 in order to decide what to do in the following instruction cycle that starts in state 0. Also, interrupt exception, DMA, or instruction processing decisions are based on the highest priority activity concurrently being requested. Interrupt exceptions are the highest priority, and instruction processing is the lowest. If no interrupt exceptions are requested, then the processor will check to see if a DMA cycle is requested. If no DMA cycle is requested, then a normal instruction cycle is performed. When an interrupt condition is recognized, the DSP3210 saves the processor context in interrupt shadow registers, and the processor branches to the appropriate interrupt handling routine based on the source of the interrupt. When the interrupt routine executes an **ireturn** instruction, normal instruction processing is reinstated by using the context saved in the interrupt shadow registers.

Error exceptions are arbitrated on a state-by-state basis. The detection of an error condition will cause the processor to abort the instruction at its current point of execution; processor level and do-loop control information are reset, and DMA and interrupts are disabled. Table 7-1 shows the priority of exception processing. Exceptions have a fixed priority, but masking can be used to selectively enable desired exceptions. Error and interrupt masking is done in the exception mask register (emr). Instructions that lock out interrupts will not respond to an interrupt for the duration of the lock. Error exceptions can not be locked out, although some exceptions can be masked. When an error condition is recognized, the current instruction is aborted and the processor branches to the appropriate error debug handler based on the source of the error.

The DSP3210 supports a 16-entry exception vector table. Each entry is a pair of 32-bit words starting at the location specified in the exception vector table pointer (evtp) register, r22. On reset, the evtp register is cleared to zero. To branch to the exception handling routine, the processor multiplies the vector number by eight to determine the offset. It adds the offset to the address in the exception vector table pointer (evtp) to obtain the memory address of the exception vector. A branch instruction using this address is executed followed by a nop. Figure 7-5 shows the layout of the exception vector table.

Only the initial reset vector is fixed in the DSP3210's memory map (evtp = address 0 on reset). When modifying the location of the vector table, the new table entries must be established starting at the new base address. Then, the evtp pointer is changed to point to the new base address, and then interrupts are enabled in the emr register. The layout of the emr is shown in Table 7-2.

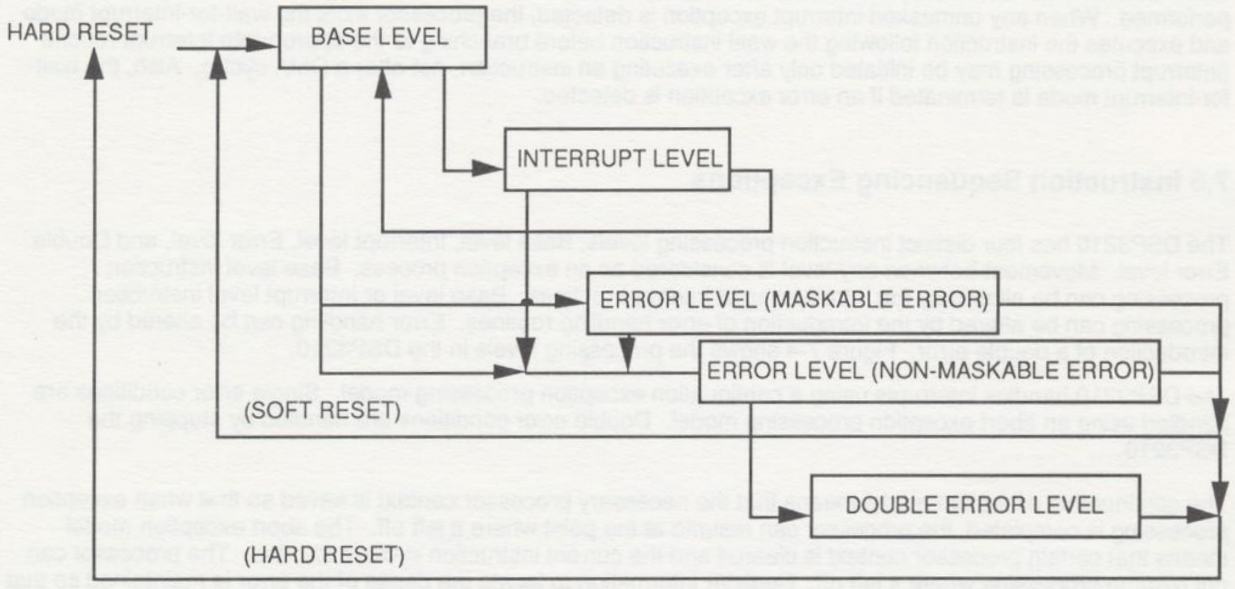


Figure 7-4 Processing Levels

Table 7-1 Priority of Exception Processing

Type	Maskable	Description
Errors		**Abort exception model**
Reset	No	Highest priority error condition
Bus Error	No	
Illegal opcode	No	
Reserved	No	
Address Error	Yes	
DAU Overflow/Underflow	Yes	
NAN	Yes	
Reserved	Yes	Lowest priority error condition
Interrupts		**Continuation exception model**
External Interrupt 0	Yes	Highest priority interrupt condition
Timer	Yes	
Reserved	Yes	
SIO Input Buffer	Yes	
SIO Output Buffer	Yes	
SIO DMA Input Frame	Yes	
SIO DMA Output Frame	Yes	
External Interrupt 1	Yes	Lowest priority interrupt condition

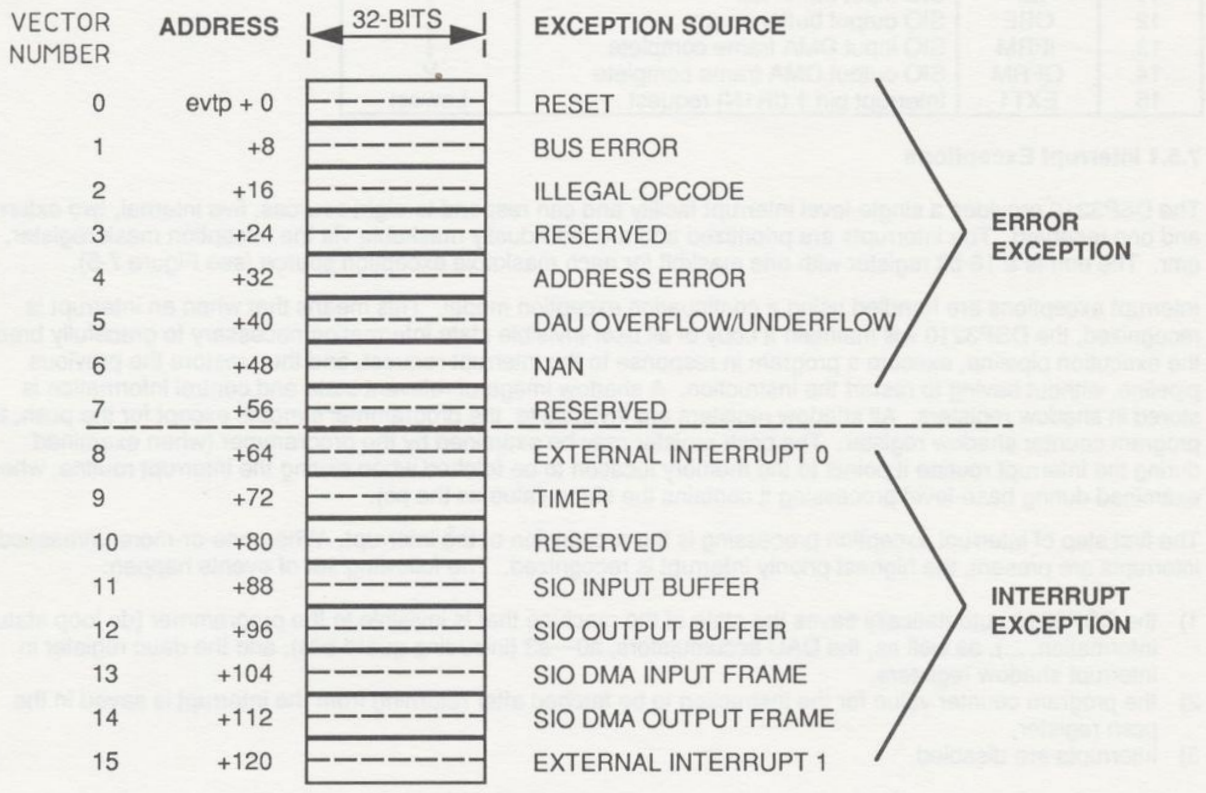


Figure 7-5 Exception Vector Table

Table 7-2 emr Register Encoding

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3 2 1	0
Field	EXT1	OFRM	IFRM	OBE	IBF	RES	TIMER	EXT0	RES	NAN	U/V	AERR	RES	SL

Each bit in this register corresponds to one of the maskable sources for an exception. A value of 1 in a position enables the corresponding interrupt source; a value of 0 disables the source. On reset all bits are set to 0 (zero).

Bit(s)	Field	Function	Priority
0	SL	If 1, select processor level to be visible on IACK[1—0]. Encoding: 00 - Base level 01 - Interrupt level 10 - Error level 11 - Double error level	—
1—3		Reserved	—
4	AERR	Addressing Error	Highest
5	U/V	DAU Underflow/Overflow	|
6	NAN	IEEE NAN	|
7		Reserved	|
8	EXT0	Interrupt pin 0 (IR0N) request	|
9	TIMER	TIMER counted to zero	|
10		Reserved	|
11	IBF	SIO input buffer full	|
12	OBE	SIO output buffer empty	|
13	IFRM	SIO input DMA frame complete	|
14	OFRM	SIO output DMA frame complete	V
15	EXT1	Interrupt pin 1 (IR1N) request	Lowest

7.5.1 Interrupt Exceptions

The DSP3210 provides a single-level interrupt facility and can respond to eight sources, five internal, two external, and one reserved. The interrupts are prioritized and are individually maskable via the exception mask register, emr. The emr is a 16-bit register with one maskbit for each maskable exception source (see Figure 7-5).

Interrupt exceptions are handled using a continuation exception model. This means that when an interrupt is recognized, the DSP3210 will maintain a copy of all user-invisible state information necessary to gracefully break the execution pipeline, execute a program in response to the interrupt request, and then restore the previous pipeline, without having to restart the instruction. A shadow image of relevant state and control information is stored in shadow registers. All shadow registers are invisible to the programmer's model except for the pcsh, the program counter shadow register. The pcsh register may be examined by the programmer (when examined during the interrupt routine it points to the memory location to be fetched when exiting the interrupt routine; when examined during base-level processing it contains the same value as the pc).

The first step of interrupt exception processing is the recognition of the interrupt. When one-or-more unmasked interrupts are present, the highest priority interrupt is recognized. The following set of events happen:

1) the DSP3210 automatically saves the state of the machine that is invisible to the programmer (do loop status information, ...), as well as, the DAU accumulators, a0—a3 (including guard bits), and the dauc register in interrupt shadow registers,
2) the program counter value for the instruction to be fetched after returning from the interrupt is saved in the pcsh register,
3) interrupts are disabled.

The second step of interrupt exception processing is branching to the appropriate interrupt handling routine. To do this the following set of events happen:

1) the exception vector is multiplied by 8 to obtain the table offset,
2) the offset is added to the address in the exception vector table pointer by the instruction **goto evtp + offset** to obtain the memory address of the exception entry and the branch is executed,
3) the goto is followed by a **nop** to account for branch latency.

Internal states that are visible to the programmer (other than the DAU accumulators, a0—a3) and used in the interrupt level must be saved and restored by the interrupt service routine.

The last step of interrupt exception processing is returning to the base processing level. This is done by execution of the ireturn instruction. The ireturn instruction:

1) restores the state of the machine that is not visible to the user, as well as a0—a3 and the dauc register,
2) fetches the instruction pointed to by the pcsh register,
3) causes the next instruction to be executed to come from the instruction shadow register (irsh),
4) re-enables interrupts.

Three instructions are not interruptible:
—CAU register load from memory or I/O register, or register load from memory
e.g. **rD = *rS** or **rD = ior1 or ior1 = *rs**
—Branch instructions, including goto, conditional goto, call and return
e.g. **goto rN**
—Stack pointer increment/decrement,
e.g. **sp = (long) sp++**

Figure 7-6 shows an example of the interrupt exception process. The row labelled IRSH represents the Instruction Shadow Register.

		I0				goto evtp + N				nop				Interrupt Service Routine		ireturn				I1			
State	3	0	1	2	3	0	1	2	3	0	1	2	3	0	3	0	1	2	3	0	1	2	3
Address Bus		I_1		X_0	Y_0		Z_{-3}			I_{i0}										I_2			
Data Bus	I_0					Z_{-3}	X_0	Y_0					I_{i0}										I_2
IR		I_0	I_0	I_0	I_0	goto evtp + p					nop					I_{iN}	I_{iN}	I_{iN}	I_{iN}	I_1	I_1	I_1	I_1
IRSH		I_0	I_0	I_0	I_0	I_1	I_1	I_1	I_1	I_1	I_1	I_1	I_1			I_1	I_1	I_1	I_1	I_1	I_1	I_1	I_1

Interrupt Arbitration and Prioritization
Interrupt Request Synchronization

Figure 7-6 Interrupt Exception Timing

7.5.1.1 Quick Interrupt

Each entry in the exception vector table may contain instructions. In general, these may be used for a goto and the delayed branch latent instruction. Alternatively, by using these two locations (and any entry pair following this that are not being used) for instructions, an interrupt routine with an overhead of three instruction cycles can be performed. An example of a quick interrupt operation performed by the DSP3210 while processing the instruction stream I1, I2, and I3 is shown below:

```
I1                  /* Base level instruction #1 */
I2                  /* Base level  instruction #2
goto evtp + offset  /* overhead branching to interrupt
nop                 /* latent branch instruction
IR1                 /* useful interrupt routine instruction
ireturn             /* no latent instruction executed after ireturn */
I3                  /* Base level  instruction #3 */
```

7.5.2 Error Exceptions

Error exceptions are handled using an abort exception model. The errors that can cause an error exception are a reset (power-on or RESTN asserted), bus error, illegal opcode, address error, DAU overflow or underflow, or the detection of a NAN on an IEEE to DSP32 format conversion. Error exception processing can be entered from the base level or from the interrupt level.

The first step of error exception processing is the recognition of the error. When one-or-more non-maskable errors are present or maskable errors are present and unmasked, the highest priority error is recognized. The following set of events happen:

1) the instruction currently executing is aborted
   a) do-loops are terminated if active,
   b) pending memory transactions are cancelled,
2) on-going memory transactions are completed (but the data is ignored),
3) DMA operations are disabled (wait-for-interrupt is accordingly terminated), and
4) interrupts are disabled.

The second step of error exception processing is to branch to the appropriate error handling routine. To branch to the exception handling routine:

1) the processor multiplies the exception vector number by eight to determine the offset,
2) the offset is added to the address in the exception vector table pointer (evtp) by the instruction **call evtp + offset (r20)** to obtain the memory address of the exception entry and the call is executed,
3) the pc is placed in r20, the error trace register, as a result of the call (the pc represents the address of the prefetched instruction),
4) the call instruction is followed by a **nop** to account for branch latency,

Figure 7-7 shows an example of the error exception process. In this example, an error is detected during state 1 of I0. I0 is aborted and the processor state is forced to state 3 for three clock cycles. This is referred to as the error state. Following the error state, the processor executes the call and nop instructions that vector it to the error handling routine. Table 7-3 shows the registers affected by error exceptions other than reset. Note that on a reset error exception, additional registers are cleared as described in 7.5.4 Reset.

The cause of the error exception is broken into two types; non-maskable errors, and maskable errors. When an error exception is taken, the type of error is recorded so that if subsequent errors occur, the processor can respond to them. If the processor is in the error level caused by a maskable error and another maskable error is detected, the error is ignored. If the processor is in the error level caused by a maskable error and a non-maskable error is detected, an error exception is taken to the non-maskable error source handler routine. If the processor is in the error level caused by a non-maskable error and another non-maskable error occurs, the processor executes a double error exception (see 7.5.3 Double Error Exceptions).

The error level processing state can be exited by a software reset of the processor. A software reset of the processor consists of 1) configuring control and data registers to their reset state or any other desired state, 2) resetting the error level to the base level via the **sftrst** instruction, and 3) branching to 0 or any other desired location. The sole function of the sftrst instruction is to change the processor level to the base level. Steps 1 and 3 are performed in software.

7.5.2.1 Double Error Exceptions

If during error exception processing of a maskable error or non-maskable error another maskable error is detected, the processor will ignore it. If during error exception processing of a maskable error a non-maskable error is detected, the processor will initiate another error exception. If during error exception processing of a non-maskable error another non-maskable error is detected, the processor will enter a double error exception. A double error exception will cause the processor to enter a state that can only be left by asserting the RESTN pin or powering down and powering up the DSP3210. The double error exception is signalled by the simultaneous assertion of IACK0 and IACK1.

Table 7-3 Registers Affected by Error Exceptions Other Than Reset

Register	Value	Comments
r20	Previous PC	Result of call evtp + VN*8 (r20).*
pc	evtp + EN*8	Result of call evtp + VN*8 (r20).
tcon[0]	0	Timer decrementing is disabled.
dmac[0,4]	0	SIO DMA is disabled.
pcw[13]	0	Enable writing of pcw register.

* VN is the exception vector number.

		I0			Error State			call evtp + N (20)				nop						
State	3	0	1	2	3	3	3	0	1	2	3	0	1	2	3	0	1	2
Address Bus		I_1										I_{e0}				I_{e1}		
Data Bus	I_0				I_1										I_{e0}			
IR		I_0	I_0					call evtp + N (r20)				nop				I_{e0}	I_{e0}	

Error Exception Detected

Figure 7-7 Example of Error Exception Process

7.5.3 Description of Error Exception Sources

The next sections describe each error exception in detail. Reset is a special case of the error exception and is discussed in Section 7.5.4 Reset.

7.5.3.1 Bus Error

A bus error occurs when an external memory transaction is terminated with BERRN asserted and SRDYN deasserted. When this code is detected, the current bus cycle is terminated and outstanding fetches are invalidated (see 6.4 Bus Error).

7.5.3.2 Illegal Opcode

The opcode of the instruction is defined as the upper 6-bits of the instruction (31—26). Seven of the 64 possible opcodes are defined to be illegal.

	Bits 31—26 Binary	Bits 31—26 Hex
1	0b000000	0x00
2	0b000001	0x01
3	0b000010	0x02
4	0b001111	0x0F
5	0b010110	0x16
6	0b010111	0x17
7	0b100010	0x22

7.5.3.3 Address Error

An address error occurs when a 32-bit word is addressed with an address not divisible by 4, or a 16-bit half-word is addressed with an address not divisible by 2.

7.5.3.4 DAU Underflow/Overflow

Overflow occurs if a DAU operation results in a number that exceeds the range of positive or negative numbers that are representable in the 40-bit format. An overflow occurs if the result is more positive than $2 - 2^{-31} * 2^{127}$ or more negative than $-2 * 2^{127}$. Underflow occurs if a DAU operation results in a number that is smaller than the range of positive or negative numbers that are representable in the 40-bit format. An underflow occurs if the non-zero result is more negative than $1 * 2^{-127}$ and more positive than $-[1 + 2^{-31}]*2^{-127}$.

7.5.3.5 NAN

A NAN is detected in the IEEE to DSP32 floating-point conversion operation if the IEEE number is a NAN (e = 255 and f ≠ 0 and s = don't care).

7.5.4 Reset

The DSP3210 provides two reset mechanisms, a power-on reset circuit and an external reset pin, RESTN. The following terminology is needed to define reset operation:

Power-on reset: The power-on reset circuit is active for the first 8 rising clock edges after power has been applied ($V_{DD} \geq 3$ V) to the device and it puts the chip in the reset state. After 8 clock cycles, the logic level of RESTN controls DSP3210 operation. If RESTN is low, the DSP3210 remains in the reset state. If RESTN is high, the DSP3210 performs the reset sequence, and enters the base level processing state.

Reset state: The DSP3210 is in the reset state when the internal reset signal is asserted, initializing the internal states of the chip. In the reset state, external memory transactions, bus requests, wait-for-interrupt, and error or interrupt exception processing are immediately terminated. Internal registers that are affected during reset are shown in Table 7-4. This state is entered when either power-on reset is detected, or RESTN is asserted.

Reset Sequence: The reset sequence is the same as step two of the error exception process (see 7.5.1 Error Exceptions), and it consists of executing two internally generated instructions:

(1) **call evtp + 0 (r20)**
(2) **nop**.

Table 7-4 Registers Affected by Reset

Register	Value	Comments
evtp	0	Exception base address set to 0
r20	Previous PC	Result of call evtp + 0 (r20)
pc	0	Result of call evtp + 0 (r20)
dauc[7—0]	0	Round, etc.
pcw[9—0]	0x38f	Maximum waits, 256 word page, etc.
pcw[10]	bio[7]	Set to logic state applied to bio[7]
pcw[11]	bio[6]	Set to logic state applied to bio[6]
pcw[12]	bio[5]	Set to logic state applied to bio[5]
pcw[13]	bio[4]	Set to logic state applied to bio[4]
pcw[15—14]	0b00	Block Move
emr[15—0]	0	All maskable exceptions masked
tcon[7—0]	0	Timer disabled
timer[31—0]	0xffffffff	Maximum value
ioc[31—0]	0	Passive mode, 32-bit data length, etc.
dmac[7—0]	0	SIO DMA disabled
bioc[7—0]	0	BIO pins configured as inputs
bio[1,3,5,7,9,11,13,15]	0	BIO output register bits

7.5.4.1 Reset Timing

A description of the reset pin follows:

RESTN **Reset** (Open Drain; Active Low). The RESTN pin is an open-drain signal. When the power-on reset circuit is active, the DSP3210 asserts RESTN by pulling it down. The external system may also pull RESTN pin low at the same time or at other times to cause the DSP3210 to enter the reset state. RESTN is an asynchronous input that is internally synchronized to the falling edge of CKI. On a zero-to-one transition of RESTN, the reset sequence is initiated. The rising edge of RESTN is used to latch the signal values on pins bio[7—4] into pcw[10—13], respectively.

On power-on of the DSP3210, on-chip circuitry puts the device in the reset state for the first 8 clock cycles (the first 8 rising clock edges of CKI). At this time, the logic level of RESTN controls the DSP3210's operation. Figure 7-9a shows the timing for power-on reset. Note that all bus interface signals are placed in the high-impedance state during the reset state and for the first 11 clocks periods of the reset sequence. Outputs and Biputs, other than the BIO and the Bus Interface signals, are placed in the high-impedance state while the power-on circuit is active only. No RESTN pulse is required to reset and start the DSP3210 running when using the power-on capability. The RESTN pin must be pulled to V_{DD} through an external resistor.

An externally controlled reset can be initiated at any time. RESTN is an asynchronous input that is internally synchronized to the falling edge of CKI. Figure 7-9b shows the timing for externally controlled reset operations. The minimum pulse width for RESTN is 4 periods of CKI. All BIO pins are immediately put in the high-impedance state so that signal values may be applied on the rising edge of RESTN. Outputs and Biputs other than the BIO and the Bus Interface signals may be placed in the high-impedance state by asserting ZN.

For both power-on reset and externally controlled reset, the BIO[4—7] inputs must be valid prior to the rising edge of RESTN and held following it in order to guarantee correct operation and to initialize pcw[13—10]. The rise time of RESTN between 0.4 V and 2.4 V must be less than 4 ns. With up to 50 pF of external load, a 1K Ω resistor can be used.

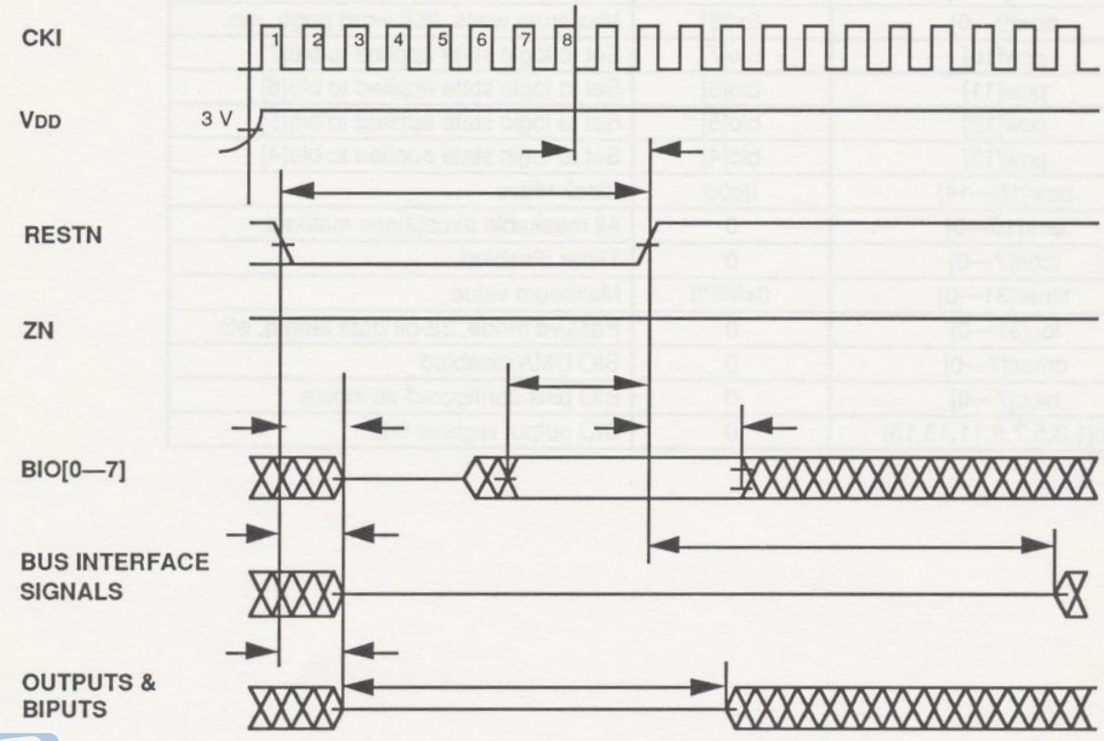


Figure 7-8A Power-On Initiates the Reset Sequence

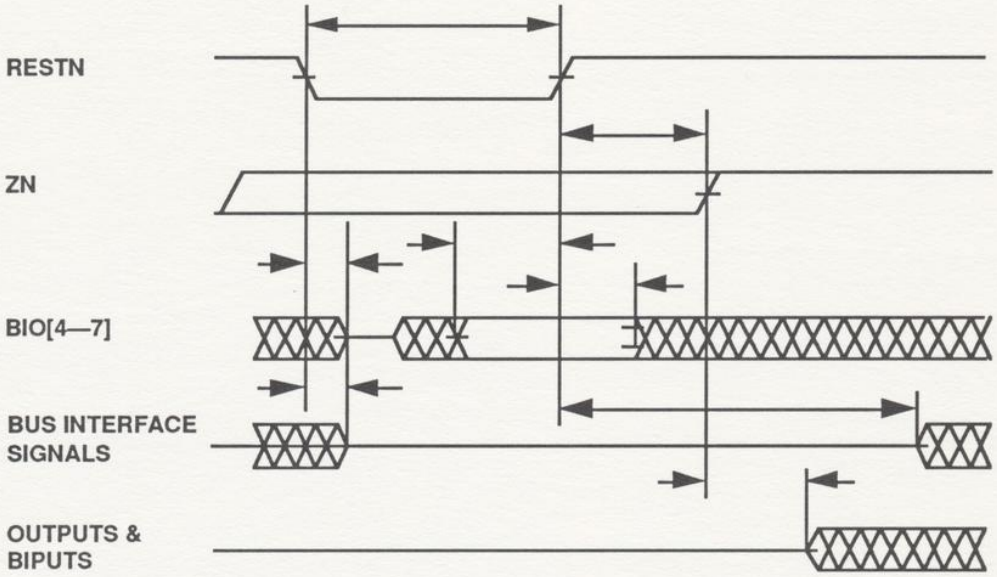


Figure 7-8B RESTN Initiates the Reset Sequence


Smile

Chapter 8

Data Processing Functional Units

CHAPTER 8. DATA PROCESSING FUNCTIONAL UNITS

8. DATA PROCESSING FUNCTIONAL UNITS

The DSP3210 provides two data processing units; Control Arithmetic Unit (CAU) and Data Arithmetic Unit (DAU). The CAU performs integer arithmetic, and the DAU performs floating-point arithmetic and data type conversions. Detailed descriptions of these data processing units are contained in this section.

8.1 Control Arithmetic Unit (CAU)

The CAU is responsible for performing address calculations, branching control, and 16- or 32-bit integer arithmetic and logic operations. It is a RISC-like processor consisting of a 32-bit arithmetic logic unit (ALU), 32-bit Barrel Shifter, and a 32-bit wide register stack.

8.1.1 CAU Programming Model

The CAU is used in the execution of both classes of instructions: CA (Control Arithmetic) and DA (Data Arithmetic). CA instructions perform load/store, branching control, and 16- and 32-bit integer arithmetic and logical operations. CA instructions are performed solely by the CAU. DA instructions use the DAU to perform floating-point multiply/accumulate and special function operations and the CAU to generate addresses for the operands. Each DA instructions can have up to three operand memory accesses per instruction, and the CAU is responsible for generating the addresses for these operands. For a detailed description of the instruction set, see Chapter 4 Instruction Set.

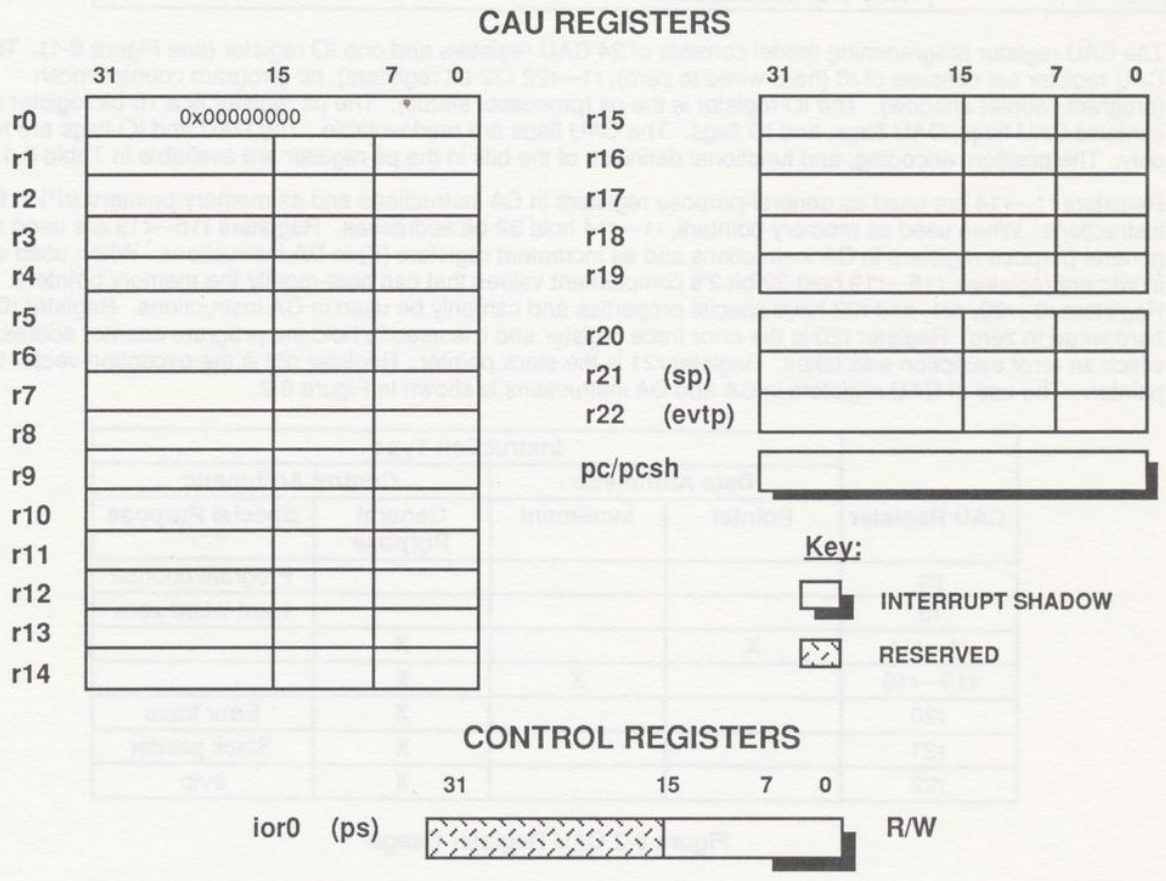


Figure 8-1 CAU Programming Model

Table 8-1 ps Register

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Field	0	0	ir1	ir0	fb	sy	obe	ibf	V	U	Z	N	c	v	z	n

Bit(s)	Field	Function
0	n	CAU **n** flag; means that the ALU result is negative.
1	z	CAU **z** flag; means that the ALU result is zero.
2	v	CAU **v** flag; means that the ALU result overflowed.
3	c	CAU **c** flag; means that the ALU result had a carry out of the MSB.
4	N	DAU **N** flag; means that the DAU result is negative.
5	Z	DAU **Z** flag; means that the DAU result is zero.
6	U	DAU **U** flag; means that the DAU result underflowed.
7	V	DAU **V** flag; means that the DAU result overflowed.
8	ibf	Serial input buffer full flag; means that the serial input buffer is full.
9	obe	Serial output buffer empty flag; means that the serial output buffer is empty.
10	sy	Serial sync set flag; means that the SY pin is high (logic one).
11	fb	Serial frame boundary flag; means that the SIO is at a frame boundary.
12	ir0	Pin IR0N is high (logic one).
13	ir1	Pin IR1N is high (logic one).
15—14	-	Reserved - read as zero.

The CAU register programming model consists of 24 CAU registers and one IO register (see Figure 8-1). The CAU register set consists of r0 (hard-wired to zero), r1—r22 (32-bit registers), pc (program counter)/pcsh (program counter shadow). The IO register is the ps (processor status). The ps register is a 16-bit register and contains CAU flags, DAU flags, and IO flags. The CAU flags are read/writable. The DAU and IO flags are read-only. The position, encoding, and functional definition of the bits in the ps register are available in Table 8-1.

Registers r1—r14 are used as general-purpose registers in CA instructions and as memory pointers (rP) in DA instructions. When used as memory pointers, r1—r14 hold 32-bit addresses. Registers r15—r19 are used as general-purpose registers in CA instructions and as increment registers (rI) in DA instructions. When used as increment registers, r15—r19 hold 32-bit 2's complement values that can post-modify the memory pointers. Registers r0 , r20, r21, and r22 have special properties and can only be used in CA instructions. Register r0 is hard-wired to zero. Register r20 is the error trace register and it is used to hold the program counter address at which an error exception was taken. Register r21 is the stack pointer. Register r22 is the exception vector table pointer. The use of CAU registers in CA and DA instructions is shown in Figure 8-2.

CAU Register	Instruction Type			
	Data Arithmetic		Control Arithmetic	
	Pointer	Increment	General Purpose	Special Purpose
pc				Program counter
r0				Hard-wired zero
r1—r14	X		X	
r15—r19		X	X	
r20			X	Error trace
r21			X	Stack pointer
r22			X	evtp

Figure 8-2 CAU Register Usage

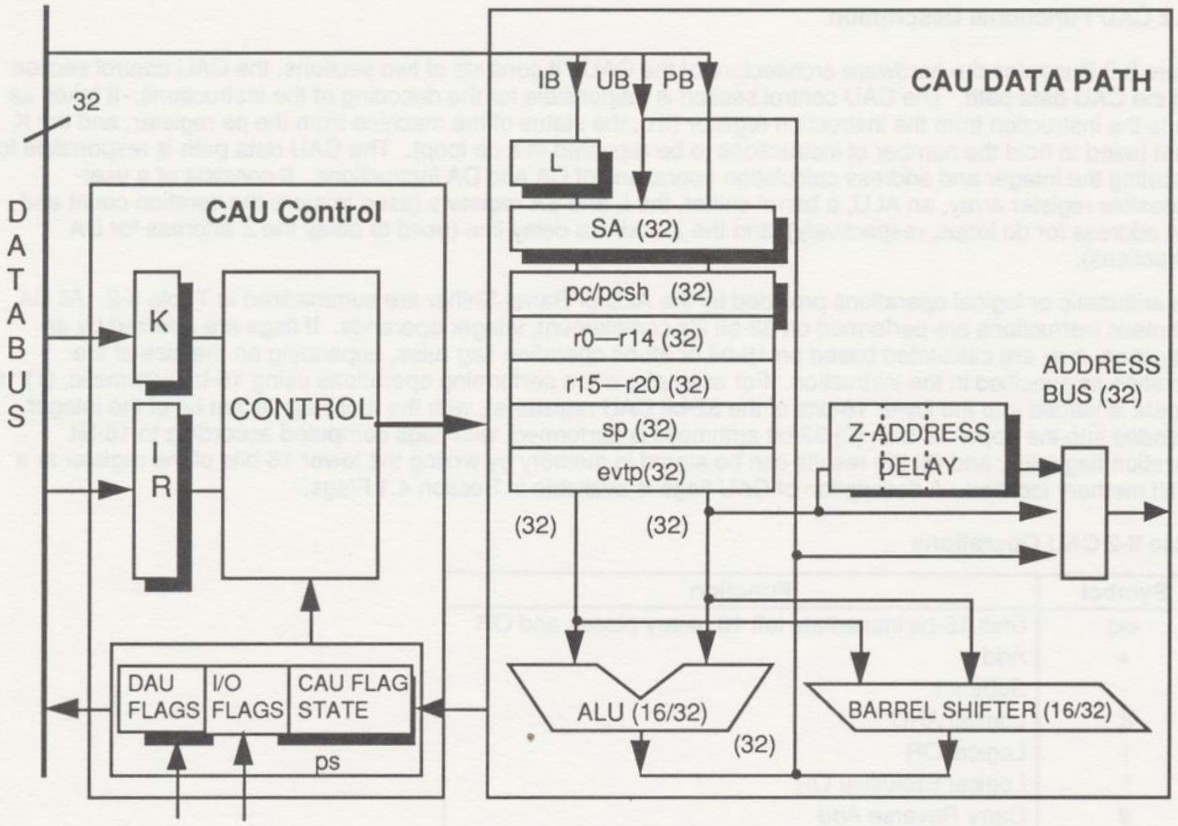
8.1.2 CAU Functional Description

Figure 8-3 illustrates the hardware architecture of the CAU. It consists of two sections, the CAU control section and the CAU data path. The CAU control section is responsible for the decoding of the instructions. It takes as inputs the instruction from the instruction register (IR), the status of the machine from the ps register, and the K count (used to hold the number of instructions to be repeated in a do loop). The CAU data path is responsible for executing the integer and address calculation operations of CA and DA instructions. It consists of a user-accessible register array, an ALU, a barrel shifter, the L and SA registers (used to store the iteration count and start address for do loops, respectively), and the Z-address delay line (used to delay the Z address for DA instructions).

The arithmetic or logical operations provided by the ALU or Barrel Shifter are summarized in Table 8-2. All CA arithmetic instructions are performed on 32-bit 2's complement, integer operands. If flags are affected by an instruction, they are calculated based on 16-bit or 32-bit operation flag rules, depending on the size of the operation as specified in the instruction. For example, when performing operations using 16-bit arithmetic, (1) 16-bit data is loaded into the lower 16-bits of the 32-bit CAU register(s), with the most significant bit of the integer extended into the upper 16 bits; (2) 32-bit arithmetic is performed, with flags computed according to 16-bit operation flag rules; and (3) the results can be stored in memory by writing the lower 16-bits of the register to a 16-bit memory location. A description of CAU flags is available in Section 4.1 Flags.

Table 8-2 CAU Operations

Symbol	Function
<<|	Shift 16-bit immediate left 16 binary places and OR
+	Add
-	Subtract
&	Logical AND
|	Logical OR
^	Logical Exclusive OR
#	Carry Reverse Add
<<	Logical Shift Left
>>	Logical Shift Right
$>>	Arithmetic Shift Right
/2	Arithmetic right shift by 1
*2	Multiply by 2
<<<1	Rotate Left through Carry
>>>1	Rotate Right through Carry
{+,-}1	Increment/Decrement



LEGEND:

IR	Instruction Register	r0	Zero
K	K counter register	r1—r14	CAU general-purpose register, DAU pointer register
L	L counter register		
SA	Start Address	r15—r19	CAU general-purpose register, DAU increment register
IB	Increment Bus		
UB	Update Bus	r20	CAU general-purpose, Error trace
PB	Pointer Bus	sp (r21)	Stack pointer
		evtp(r22)	Exception vector table pointer
		ps	Processor Status

Registers in upper case are not user accessible. Shadowed during interrupts

Figure 8-3 Block Diagram of the CAU

8.1.3 CAU Internal Operation

The internal operation of the CAU is best understood by describing the operation of its instruction set. The CA instructions consist of three types: arithmetic/logical, data move, and control instructions. Arithmetic/logical instructions are performed as register to register operations. Many of the CA arithmetic instructions can be performed using three different registers: two source registers (rS1, rS2) and one destination register (rD). CA instructions may also be conditionally executed based on flags established in previous instruction cycles. Data move instructions specify 8-, 16-, or 32-bit data transfers between CAU registers and memory, CAU register and I/O registers, and I/O registers and memory. Control instructions alter the program flow by modifying the pc (program counter).

Figure 8-4 illustrates the CAU pipeline for executing two instructions of the form:

rD – rS
if(eq) rD = rS1 + rS2

The first instruction subtracts the contents of the source register (rS) from the destination register (rD), to set/clear the CAU zero (z) flag. The second instruction checks the z flag, and if it is set (i.e., the result of the previous instruction was zero), adds the contents of rS1 and rS2, and stores the result in rD. If the flag is not set, the contents of rD are unaffected by this instruction.

During execution of DA instructions, the CAU generates up to four addresses, one address in each of the four states of an instruction cycle. In each state, the CAU can add the contents of two registers: a pointer selected from r1—r14 (placed on the pointer bus) and an increment selected from r15—r19 or a fixed increment value (placed on the increment bus). The CAU employs a 3-stage pipeline to (1) fetch from a register(s), (2) operate on the fetched operand(s), and (3) store the result in a register. Because of this pipelining, the preceding pointer is updated (result placed on the update bus), while the next pointer and increment are being accessed.

Figure 8-5 shows the CAU pipeline for executing a DA instruction of the form:

$$Z = aN = aM + Y * X$$

Since the result of the multiply-accumulate operation is not available on the data bus until state 0 of instruction cycle I_5 and the destination address is calculated during state 1 of instruction cycle I_2, the address is latched and delayed for three instruction cycles before being placed on the address bus. This three instruction cycle delay is performed by the Z-address delay block shown in Figure 8-3.

Except for the register load from memory and branch operations that have a latency of one instruction cycle, execution of a CA instruction is completed by the CAU before the next CA instruction begins execution. This simplifies use of the CAU for logic and control operations.

		Cycle 0				Cycle 1			
State	3	0	1	2	3	0	1	2	3
Pointer Bus	pc_1		rD		pc_2		rS1		pc_3
Increment Bus	+4		rS		+4		rS2		+4
ALU		+				+			
Update Bus			pc_2				pc_3		?rD
Address Bus		pc_1				pc_2			
Data Bus	I_0				I_1				I_2
IR		I_0	I_0	I_0	I_0	I_1	I_1	I_1	I_1
Flags					z				?

For instructions in the form of:

I_0: rD-rS

I_1: if(eq) rD = rS1 + rS2

where rD = destination register
rS, rS1, rS2 = source register
eq = CAU condition equal to zero (z flag)
? = modification of register and flag is conditional
pc = program counter
+ = additional operation
− = subtraction operation

Figure 8-4 CAU Internal Pipeline Timing - CA Instruction

		Cycle 0				Cycle 1				Cycle 2, 3	Cycle 4			
State	3	0	1	2	3	0	1	2	3		0	1	2	3
Pointer Bus	pc_1		rP_x	rP_y	pc_2	rP_z			pc_3					
Increment Bus	+4		rS	rI_y	+4	rI_z			+4					
ALU		+		+	+	+	+							
Update Bus			pc_2		rP_x	rP_y	pc_3	rP_z						
Address Bus		pc_1		X	Y	pc_2						Z		
Data Bus	I_0				I_1		X	Y	I_2		Z			
IR		I_0	I_0	I_0	I_0	I_1	I_1	I_1	I_1		I_4	I_4	I_4	I_4

For instructions in the form of:

I_0: $*rP_z{+}{+}rI_z = aN = aM + *rP_y{+}{+}rI_y * *rP_x{+}{+}rI_x$

where rP_z, rP_y, rP_x = pointer register for DAU - Z, Y, and X operands
rI_z, rI_y, rI_x = increment register for DAU - Z, Y, X operands
pc= program counter
+= additional operation

Figure 8-5 CAU Internal Pipeline Timing - DA Instruction

8.2 Data Arithmetic Unit (DAU)

The DAU is the primary execution unit for signal processing algorithms. This unit contains a 32-bit floating-point multiplier, a 40-bit floating-point adder, four 40-bit accumulators, Data Type conversion hardware, and two user-accessible registers, dauc (DAU control register) and ctr (Clip-Test register). The multiplier and adder work in parallel to perform 16.7 million computations per second of the form (a = b + c * d). The multiplier and adder each produce one result per instruction cycle. The DAU contains a four stage pipeline: Fetch, Multiply, Add, and Store. Thus, in any instruction cycle, the DAU may be processing four instructions, each in a different stage of execution.

8.2.1 DAU Programming Model

The register programming model for the DAU consists of 4 DAU registers, called accumulators, and two IO registers, the dauc, DAU control register and the ctr, clip-test register. Figure 8-6 shows the register programming model for the DAU. Note that all registers in the DAU are shadowed during interrupts. Each accumulator is a 40-bit register consisting of a 32-bit mantissa and an 8-bit exponent.

The dauc is used to select the 8-bit data type conversion performed for the instructions **ic** and **oc**, the truncation mode to be used when converting floating-point data to integer, and how writes are handled in conditional DA instructions, **aeq**, **alt**, and **agt**. Linear byte conversion, μ-law, and A-law share the same instruction encoding (Z=aN=ic(Y);) and are therefore distinguished only by the dauc setting. Choosing linear byte input and/or output mode overrides the corresponding μ-law/A-law setting. Loading the dauc register from memory affects the following instruction. The meaning for the settings of the dauc register are given in Table 8-3. The dauc register is cleared on reset.

The ctr stores the last six values for the DAU **N** flag. It is used for 2-D and 3-D clip-test algorithms to compute the outcode. This outcode can then be read into a CAU register for further operations. The meaning for each bit of the ctr register is given in Table 8-4.

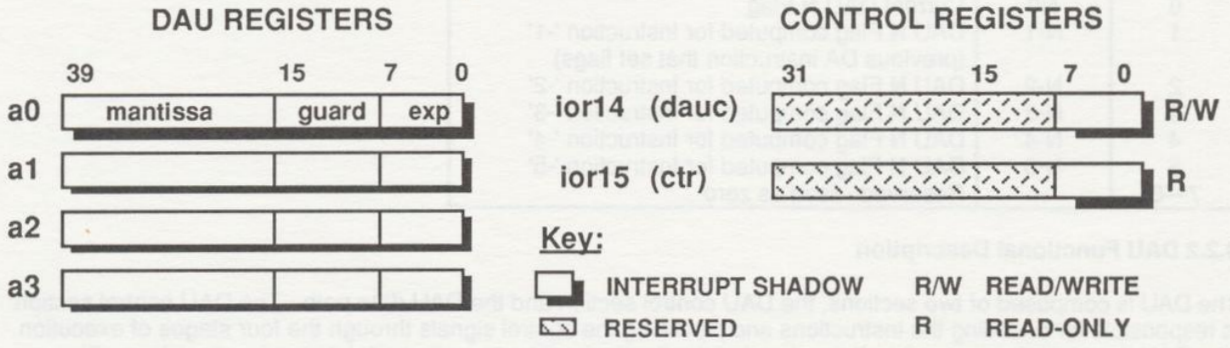


Figure 8-6 DAU Programming Model

Table 8-3 dauc Register

Bit	7	6	5 4	3 2 1 0
Field	0	COND	ROUND	CONV

Bit(s)	Field	Encoding	Function
3—0	CONV	x0x0	μ-law input conversion.
		x0x1	A-law input conversion.
		x1xx	Unsigned linear byte input conversion.
		0x0x	μ-law output conversion.
		0x1x	A-law output conversion.
		1xxx	Unsigned linear byte output conversion.
5—4	ROUND	x0	Round-to-nearest on float-to-integer conversions.
		01	Truncate towards -∞ on float-to-integer conversion.
		11	Truncate towards 0 on float-to-integer conversion.
6	COND	0	DA instructions *ifalt*, *ifaeq*, and *ifagt* have unconditional Z-writes.
		1	DA instructions *ifalt*, *ifaeq*, and *ifagt* have conditional Z-writes.
7	0	0	Must be set to zero; Read as zero.

Table 8-4 ctr Register

Bit	7	6	5	4	3	2	1	0
Field	0	0	N-5	N-4	N-3	N-2	N-1	N0

Bit(s)	Field	Meaning
0	N0	Current DAU **N** Flag
1	N-1	DAU **N** Flag computed for Instruction '-1' (previous DA instruction that set flags)
2	N-2	DAU **N** Flag computed for Instruction '-2'
3	N-3	DAU **N** Flag computed for Instruction '-3'
4	N-4	DAU **N** Flag computed for Instruction '-4'
5	N-5	DAU **N** Flag computed for Instruction '-5'
7—6	-	Reserved - read as zero

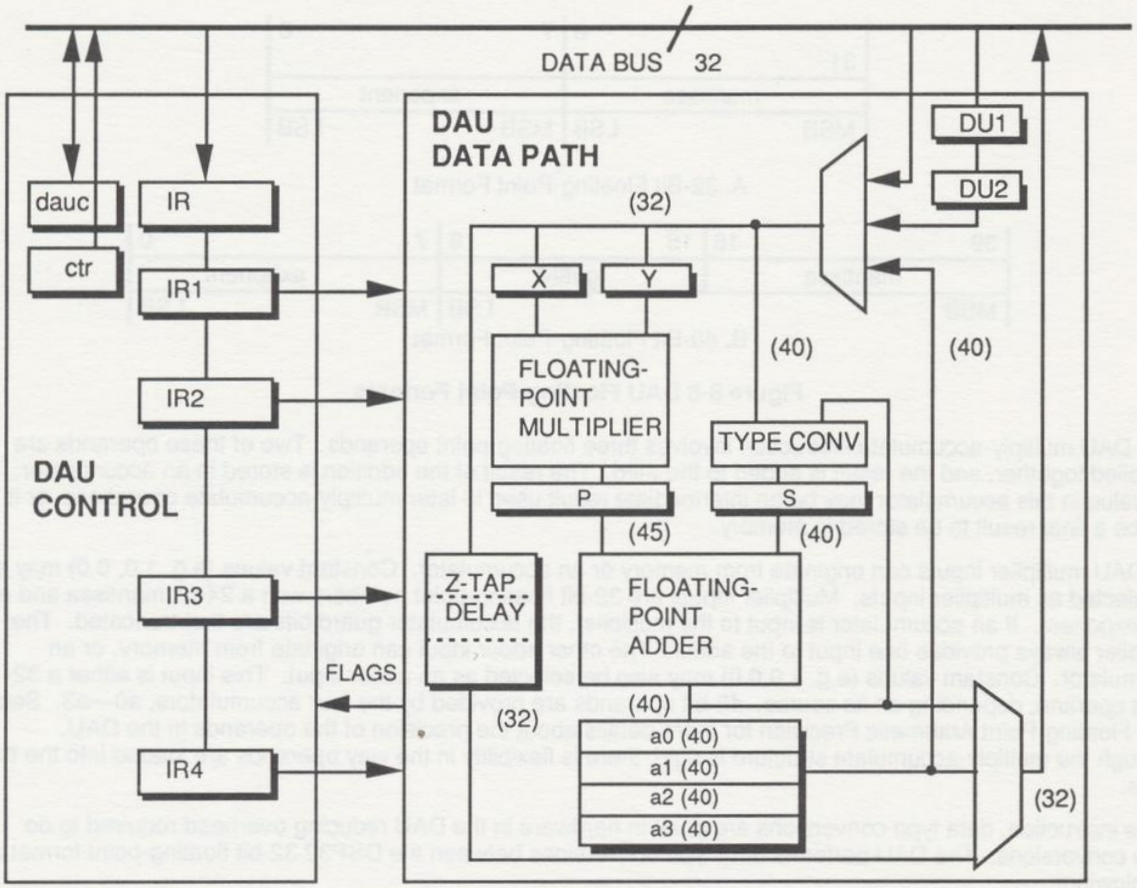
8.2.2 DAU Functional Description

The DAU is composed of two sections, the DAU control section and the DAU data path. The DAU control section is responsible for decoding the instructions and pipelining the control signals through the four stages of execution. The DAU data path is responsible for performing the floating-point arithmetic and data type conversions. Figure 8-7 illustrates the hardware architecture of the DAU.

The DAU supports two floating-point formats, single precision (32-bit) and extended single precision (40-bit). Extended single precision provides 8 additional mantissa guard bits. The four 40-bit accumulators, a0—a3, reduce round-off problems to ensure 24-bit precision. Post-normalization logic transparently shifts binary points and adjusts exponents to prevent inaccurate rounding of bits when the floating-point numbers are added or multiplied, eliminating concerns like scaling and quantization error. All normalization is done automatically, so the result in the accumulator is fully normalized.

The DAU contains both 32-bit and 40-bit registers and buses to support two floating-point formats, 32-bit and 40-bit extended precision floating-point. The 40-bit format provides 8 additional mantissa guard bits used by the DAU in accumulate operations. Figure 8-8 shows the two floating-point formats.

When a 40-bit bus connects to a 32-bit register or multiplexer, the guard bits are excluded (truncated). For example, when writing a 40-bit accumulated result to memory, the result is first truncated to 32-bits. When a 32-bit register drives a 40-bit bus, the guard bits (bits 15—8) are zeroed on the 40-bit bus.

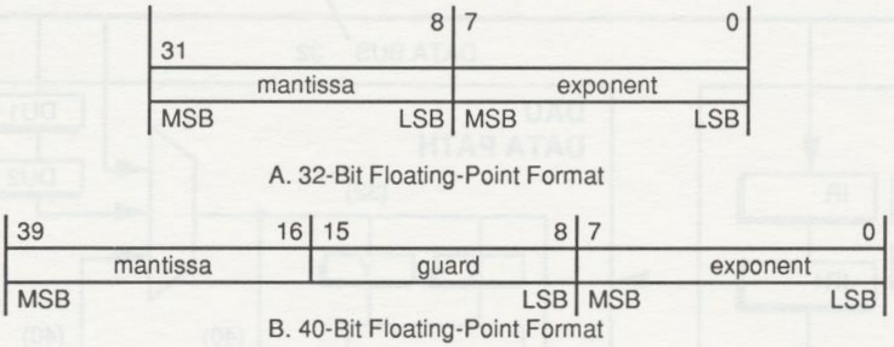


LEGEND:

IR	Instruction Register	X	Multiplier input register
IR1	IR for instr. cycle 1	Y	Multiplier input register
IR2	IR for instr. cycle 2	P	Product register
IR3	IR for instr. cycle 3	S	Special function register
IR4	IR for instr. cycle 4	dauc	DAU Control Register
DU1,DU2	Input operand delay registers	ctr	Clip-test Register
		a0—a3	Accumulators 0—3

Registers in upper case are not user accessible. ▭ Shadowed during interrupts.

Figure 8-7 Block Diagram of the DAU



A. 32-Bit Floating-Point Format

B. 40-Bit Floating-Point Format

Figure 8-8 DAU Floating-Point Formats

Each DAU multiply-accumulate instruction involves three floating-point operands. Two of these operands are multiplied together, and the result is added to the third. The result of the addition is stored in an accumulator. The value in this accumulator may be an intermediate result used in later multiply-accumulate operations, or it may be a final result to be stored in memory.

The DAU multiplier inputs can originate from memory or an accumulator. Constant values (e.g. 1.0, 0.0) may also be selected as multiplier inputs. Multiplier inputs are 32-bit floating-point numbers with a 24-bit mantissa and an 8-bit exponent. If an accumulator is input to the multiplier, the accumulator guard bits are first truncated. The multiplier always provides one input to the adder. The other adder input can originate from memory, or an accumulator. Constant values (e.g. 1.0,0.0) may also be selected as an adder input. This input is either a 32- or 40-bit operand, depending on its source. 40-bit operands are provided by the four accumulators, a0—a3. See 8.2.3 Floating-Point Arithmetic Precision for more details about the precision of the operands in the DAU. Although the multiply-accumulate structure is rigid, there is flexibility in the way operands are loaded into the three inputs.

Single instruction, data type conversions are done in hardware in the DAU reducing overhead required to do these conversions. The DAU performs data type conversions between the DSP32 32-bit floating-point format and the folowing:

—IEEE P754 standard 32-bit floating-point
—16- and 32-bit integer
—8-bit u-law and A-law formats
—8-bit unsigned integer

The DAU also provides an instruction to convert a 32-bit floating-point operand to a 3-bit seed value used for reciprocal approximation in division operations.

8.2.3 Floating-Point Arithmetic Precision

Each DAU multiply-accumulate instruction (of the form A = S + Y * X) involves three floating-point operands. Two of these operands (X and Y) are multiplied together, and the result (P) is added to the third (S). The result of the addition is stored in one of four accumulators (a0—a3). The contents of this accumulator may be an intermediate result used in later multiply-accumulate operations, or a final result to be stored in memory.

Figure 8-9 illustrates the precision of the floating-point operands and results involved in DAU multiply-accumulate operations. (Because precision is a function of the mantissa length, discussion of the exponent is omitted here.) An implied binary point is included in the figure. Bits to the right of the binary point constitute the fractional part of the number, while those to the left constitute the integer part.

Recall that the magnitude of the mantissa (M), in the multiplier inputs (X and Y), adder input (S) and adder result (a0—a3) is always normalized such that $-2 \le M < -1$ or $1 \le M < 2$. Thus, the leading 1 (positive mantissa) or leading 0 (negative mantissa) does not appear explicitly in the floating-point word, and is called an **implicit** bit.

The result of a multiplication (P) could have a denormalized mantissa (e.g. in cases such as $-2 * -2 = 4$ and $-1 * 1 = -1$), and thus three integer bits are retained in P. Normalization is done automatically in the adder, so the accumulated result is fully normalized. The adder inputs (P and S) and result (a0—a3) contain eight mantissa guard bits in addition to the standard 24 fractional bits in the mantissas.

If the result of a multiply-accumulate operation is written to memory, its guard bits are truncated. If an accumulator is input to the multiplier, its guard bits are also truncated. There is an option to first invoke the round instruction on the accumulator containing the result. The guard bits are useful when the accumulator is input to the adder. Here, the full width accumulator, including guard bits, is used.

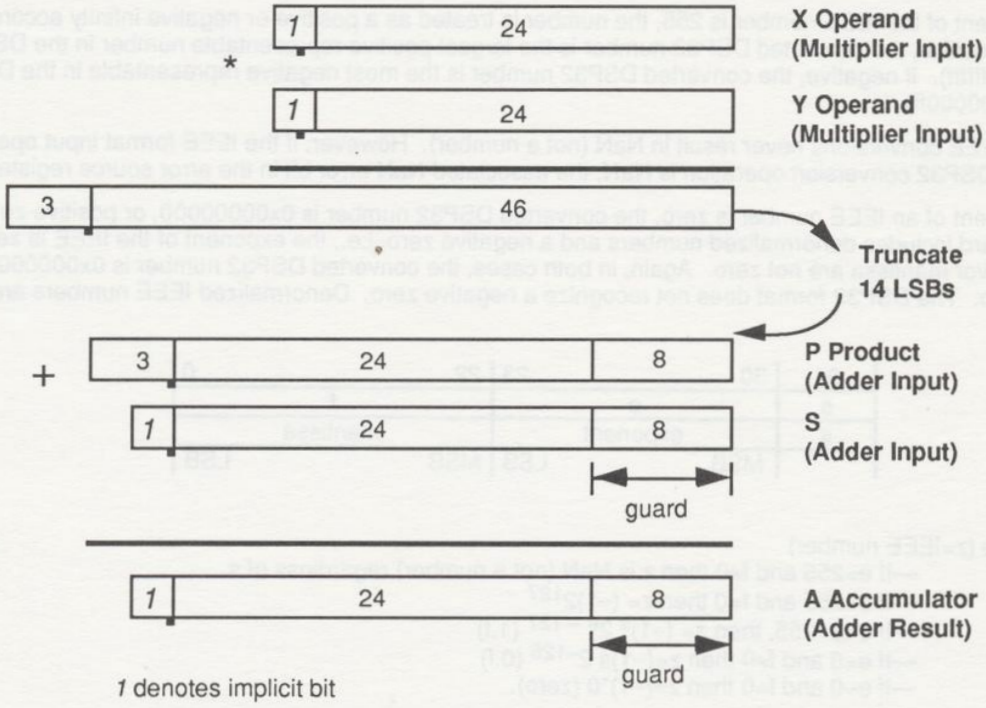


Figure 8-9 Precision of DAU Operations (mantissas only)

8.2.4 Data Type Conversions

Several data type conversions that have traditionally added considerable software overhead are handled by the DSP3210 in hardware. The DAU provides single-cycle instructions that perform data-type conversion between the DSP32 internal 32-bit floating-point format and the IEEE P754 standard 32-bit floating-point format, the 8-bit companded μ- and A-law formats, the 8-bit unsigned linear format, and the 16- and 32-bit integer formats. The following sections describe these operations in more detail.

8.2.4.1 IEEE Standard Floating-Point

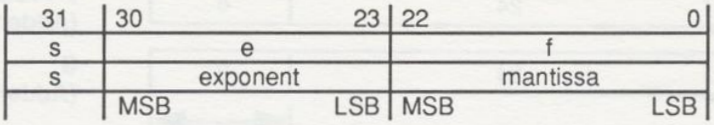
Although the DSP3210 has its own internal floating-point format, the DSP3210 is compatible with the IEEE P754 standard 32-bit floating-point format. The device contains on-chip hardware that supports format conversion to and from the IEEE format in a single instruction cycle. The single-precision IEEE standard floating-point format has a sign-magnitude mantissa and an 8-bit biased exponent. Figure 8-10 shows this format.

The floating-point format used by the DSP3210 is referred to as the DSP32 format. It is the format used by the DSP32, DSP32C, and the DSP3210. Conversions between the IEEE standard and DSP32 formats adhere to the following rules:

—If the exponent of the IEEE number is 255, the number is treated as a positive or negative infinity according to its sign. If positive, the converted DSP32 number is the largest positive representable number in the DSP32 format (0x7fffffff). If negative, the converted DSP32 number is the most negative representable in the DSP32 format (0x800000ff).

—DSP32 to IEEE conversions never result in NaN (not a number). However, if the IEEE format input operand in an IEEE to DSP32 conversion operation is NaN, the associated NaN error bit in the error source register is set.

—If the exponent of an IEEE number is zero, the converted DSP32 number is 0x00000000, or positive zero. The IEEE standard includes denormalized numbers and a negative zero, i.e., the exponent of the IEEE is zero, but the sign and/or mantissa are not zero. Again, in both cases, the converted DSP32 number is 0x00000000 or positive zero. The DSP32 format does not recognize a negative zero. Denormalized IEEE numbers are not supported.



where (z=IEEE number)

—If e=255 and f≠0 then z is NaN (not a number) regardless of s.
—If e=255 and f=0 then $z= (-1)2^{127}$
—If 0<e<255, then $z= (-1)^s 2^{e-127} (1.f)$
—If e=0 and f≠0 then $z=(-1)s 2^{-126} (0.f)$
—If e=0 and f=0 then z=(−1)*0 (zero).

Figure 8-10 IEEE Standard 32-bit Floating-Point Format

8.2.4.2 Companded

On-chip hardware supports the conversion to and from two 8-bit compressed PCM data formats: μ-law and A-law.

μ-law

The format of an 8-bit, μ-law word is as follows:

$\overline{m0}\ \overline{m1}\ \overline{m2}\ \overline{m3}\ \overline{n0}\ \overline{n1}\ \overline{n2}\ \overline{s}$

This μ-law word represents the following number:

$$Y = (-1)^S * [(16.5+M) * 2^N - 16.5]$$

where:

M = m3 m2 m1 m0 (M is 0 to 15)
N = n2 n1 n0 (N is 0 to 7)
* = multiplication
— = bar (complement)

A-law

The format of an 8-bit, A-law word is as follows:

$\overline{m0}\ m1\ \overline{m2}\ m3\ \overline{n0}\ n1\ \overline{n2}\ \overline{s}$

This A-law word represents the following number:

$$Y = (-1)^S * [(16.5+M) * 2^N] \qquad N = 1,2, \ldots, 7$$
$$Y = (-1)^S * (0.5 + M) * 2 \qquad N=0$$

where:

M = m3 m2 m1 m0 (M is 0 to 15)
N = n2 n1 n0 (N is 0 to 7)
* = multiplication
— = bar (complement)

8.2.4.3 8-bit Unsigned Integer

The DAU can perform conversions between 8-bit unsigned integers and DSP3210 floating-point values in a single instruction cycle. The 8-bit integer values are assumed to be in the range 0 to 255 when converting to floating-point. Floating-point values are converted to an 8-bit unsigned integer. If the floating-point operand exceeds the range of allowable 8-bit unsigned integers (0,255), a saturated 8-bit integer is returned (0xFF for the most positive or 0x00 for the least positive). Negative floating-point numbers are converted to 0x00. Based on the dauc register, floating-point numbers are converted to 8-bit unsigned numbers by rounding or truncation (See 8.2.5 Rounding Modes). If the 8-bit integer result is written to memory, it is loaded into an 8-bit memory location.

8.2.4.4 16-bit Integer

The DAU supports conversion from a floating-point operand to a 16-bit 2's complement integer. If the 16-bit integer result is written to memory, it is loaded into a 16-bit memory location. If the floating-point operand exceeds the range of allowable 16-bit integers (-2^{15}, $2^{15} - 1$), a saturated 16-bit integer is returned (0x8000 or 0x7fff, respectively). Based on the dauc register, floating-point numbers are converted to 16-bit linear by rounding or truncation (See 8.2.5 Rounding Modes). Similarly, the DAU supports conversion from a 16-bit integer to the equivalent 32-bit floating-point representation.

8.2.4.5 32-bit Integer

The DAU supports conversion from a floating-point operand to a 32-bit 2's complement integer. If the 32-bit integer result is written to memory, it is loaded into a 32-bit memory location. If the floating-point operand exceeds the range of allowable 32-bit integers (-2^{31}, $2^{31} - 1$), a saturated 32-bit integer is returned (0x80000000 or 0x7fffffff, respectively). Based on the dauc register, floating-point numbers are converted to 32-bit linear by rounding or truncation (See 8.2.5 Rounding Modes). Similarly, the DAU supports conversion from a 32-bit integer to the equivalent 32-bit floating-point representation.

8.2.4.6 Reciprocal Seed

In the DSP3210, a divide operation can be performed by calculating the reciprocal of the divisor, and multiplying this value by the dividend. An instruction is provided to convert, in hardware, a 32-bit floating-point number to a 32-bit seed value as a first approximation to the reciprocal of the number. This one-cycle instruction produces a seed with a mantissa accurate to 3 bits.

8.2.5 Rounding Modes

On conversions from floating-point to 32-bit or 16-bit signed integers, or to 8-bit unsigned integers, the dauc register (see Table 8-3) controls whether the integer result is rounded to nearest, truncated towards -∞, or truncated towards zero. Figure 8-11 graphically displays the differences between the three modes.

With round-to-nearest, the resulting integer is incremented if there is a one in the most significant fraction bit of the denormalized floating-point number. Note that this rounds 0.5, 1.5, 2.5,... to 1, 2, 3,..., respectively and –0.5, –1.5, –2.5,... to 0, –1, –2,..., respectively.

With truncate towards -∞, the fraction bits of the denormalized floating-point number are discarded.

With truncate towards zero, the fraction bits of the denormalized floating-point number are treated differently based on whether the number is positive or negative. For positive numbers, the fraction bits are discarded. For negative numbers, the integer is incremented by one if there were any 1s in the fraction bits.

On conversion of 8-bit unsigned integers, there is no difference between truncate towards zero and truncate towards -∞ because all negative numbers are saturated to zero.

Note that the *round()* instruction rounds a 40-bit floating-point number to a 32-bit floating-point number, and it is always performed as round-to-nearest, regardless of the dauc setting.

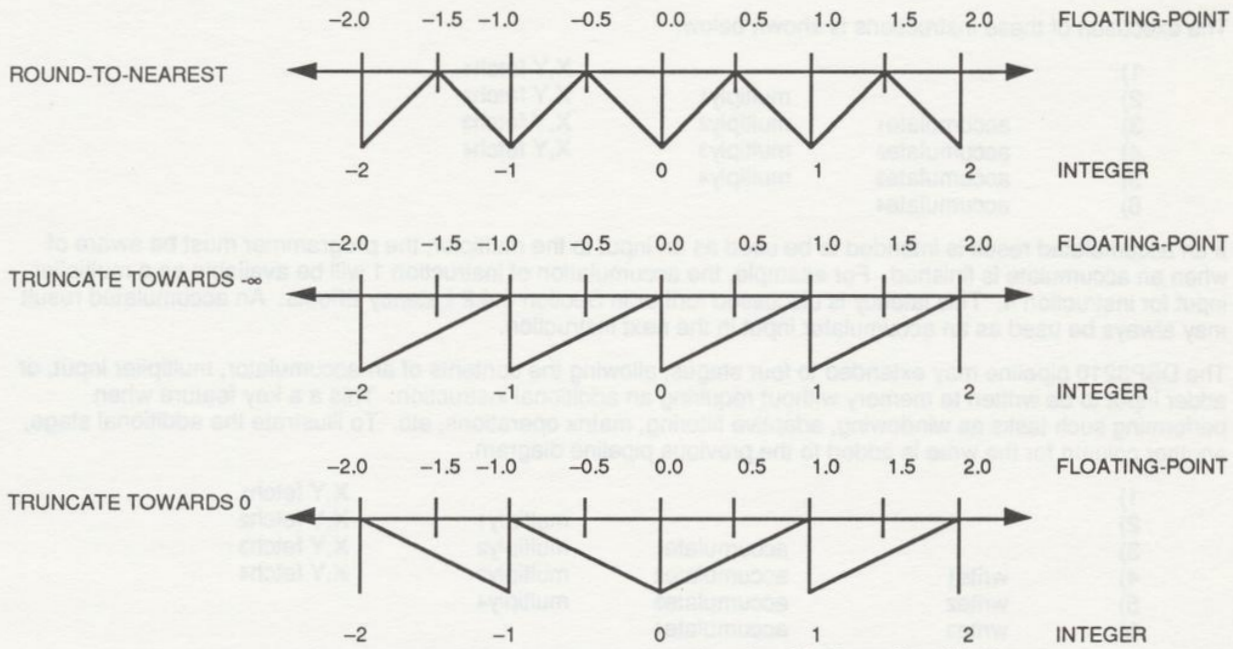


Figure 8-11 Floating-Point to Integer Mapping for All Rounding Modes

8.2.6 DAU Internal Operation

The following section presents a description of a single instruction as it passes through the DAU pipeline. The DAU supports four multiply-accumulate instruction formats, providing flexibility in choosing operand and storing results. To simplify this discussion, one format will be discussed. The DSP3210 executes the multiply-accumulate instruction

aN = aM + Y * X

in three stages (an optional fourth stage to perform a write to memory is discussed later); fetch of X and Y, multiplication of X and Y, and accumulation of the Y * X product with the contents of an accumulator, aM. This instruction executes as follows:

step 1) X,Y fetch
step 2) multiply Y * X
step 3) accumulate product with aM

If several multiply-accumulate instruction are executed one after the other, the DSP3210 automatically pipelines the instructions so that one instruction is completed every instruction cycle. This is shown in the following block of instructions where X_1-X_4 and Y_1-Y_4 are subscripted to make it easier to follow the data flow in the DAU.

1) $aN = aM + Y_1 * X_1$
2) $aN = aM + Y_2 * X_2$
3) $aN = aM + Y_3 * X_3$
4) $aN = aM + Y_4 * X_4$

The execution of these instructions is shown below:

1)			X,Y fetch1
2)		multiply1	X,Y fetch2
3)	accumulate1	multiply2	X,Y fetch3
4)	accumulate2	multiply3	X,Y fetch4
5)	accumulate3	multiply4	
6)	accumulate4		

If an accumulated result is intended to be used as an input to the multipier, the programmer must be aware of when an accumulate is finished. For example, the accumulation of instruction 1 will be available as a multiplier input for instruction 4. This latency is discussed further in Section 4.4.2 Latency Effects. An accumulated result may always be used as an accumulator input in the next instruction.

The DSP3210 pipeline may extended to four stages, allowing the contents of an accumulator, multiplier input, or adder input to be written to memory without requiring an additional instruction. This a a key feature when performing such tasks as windowing, adaptive filtering, matrix operations, etc. To illustrate the additional stage, another column for the write is added to the previous pipeline diagram.

1)				X,Y fetch1
2)			multiply1	X,Y fetch2
3)		accumulate1	multiply2	X,Y fetch3
4)	write1	accumulate2	multiply3	X,Y fetch4
5)	write2	accumulate3	multiply4	
6)	write3	accumulate4		

The syntax of such an instruction is

Z = aN = aM + Y * X

Note that the memory location written by instruction 1 will be available as multiplier input in instruction 5.

The internal pipeline timing of the DAU for a single instruction of the form Z = aN = aM + Y * X is shown in Figure 8-12. An instruction requires five instruction cycles (I_0—I_4) to be decoded and executed in the DAU. The instruction (I) appears on the data bus in state 3 preceding I_0. The X and Y registers are loaded in states 1 and 2 of I_1, respectively. The contents of these registers are multiplied during I_2 and the product is loaded into the product register, P, in state 3 of I_2. The adder input register, S, is also loaded in state 3 of I_2. The contents of the P and S registers are added during I_3 and the result is loaded into an accumulator in state 3 of I_3. This same result is then placed on the data bus (which is routed to a destination in memory) in state 0 of I_4, if a Z-field is specified in the instruction. The Z-field is optional.

The DAU supports two multiply-accumulate instruction formats called tap-update instructions. In one format,

aN = aM + (Z = Y) * X

instead of the result of the multiply-accumulate operation being written to memory, the Y operand is written. It must be written in the same cycle that the multiply-accumulate result would have been written (state 0 of I4). Because the Y operand appears on the data bus in state 2 of I1, it must be latched and held for three instruction cycles before being written to memory. This three instruction cycle delay is performed in the tap-update delay block in Figure 8-7. The other tap-update instruction has the form:

aN = (Z = Y) + X.

The four-stage pipeline in the DAU contributes to the high throughput of the DSP3210.

	Cycle 0					Cycle 1				Cycle 2				Cycle 3				Cycle 4			
State	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
RAM0					X Fetch																
RAM1						Y Fetch														Z Store	
Data Bus	I						X	Y										Z			
X							X														
Y								Y													
Multiplier										Multiply X * Y											
P													P								
S													aM								
Adder														Add S + P							
aN (a0—a3)																	aN				

For instructions of the form: Z = aN = aM + Y * X,

where X, Y = data reference (memory, a0—a3)
aM = a0-a3
aN = a0-a3
Z = data write (memory)
P= product
S= adder input

Figure 8-12 DAU Internal Pipeline Timing


Smile